Data Abstraction
1 Abstract Data Type (ADT)

2 Using an ADT

3 Examples of ADTs

4 Defining an ADT

5 Error Handling
Abstract Data Type (ADT)

An abstract data type (ADT) is one whose representation is hidden from the program that uses the ADT.

Example:
list

dsa.Counter implements java.lang.Comparable<Counter>

Counter(String id)
constructs a counter given its id

void increment()
increments this counter by 1

tally()
returns the current value of this counter

void reset()
resets this counter to zero

boolean equals(Object other)
returns true if this counter and other have the same tally, and false otherwise

String toString()
returns a string representation of this counter

int compareTo(Counter other)
returns a comparison of this counter with other by their tally.
Abstract Data Type (ADT)

An abstract data type (ADT) is one whose representation is hidden from the program that uses the ADT.
An abstract data type (ADT) is one whose representation is hidden from the program that uses the ADT.

Example

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter(String id)</td>
<td>constructs a counter given its id</td>
</tr>
<tr>
<td>void increment()</td>
<td>increments this counter by 1</td>
</tr>
<tr>
<td>int tally()</td>
<td>returns the current value of this counter</td>
</tr>
<tr>
<td>void reset()</td>
<td>resets this counter to zero</td>
</tr>
<tr>
<td>boolean equals(Object other)</td>
<td>returns true if this counter and other have the same tally, and false otherwise</td>
</tr>
<tr>
<td>String toString()</td>
<td>returns a string representation of this counter</td>
</tr>
<tr>
<td>int compareTo(Counter other)</td>
<td>returns a comparison of this counter with other by their tally</td>
</tr>
</tbody>
</table>
Abstract Data Type (ADT)

Salient features of an ADT:

• Some entries (called constructors) have the same name as the class and no return type
• Some entries (called methods) lack the static keyword and operate on data-type values
• Some methods such as equals(), hashCode(), and toString() are inherited from the parent java.lang.Object class and overridden in the ADT
Abstract Data Type (ADT)

Salient features of an ADT:

- Some entries (called constructors) have the same name as the class and no return type.
- Some entries (called methods) lack the `static` keyword and operate on data-type values.
- Some methods such as `equals()`, `hashCode()`, and `toString()` are inherited from the parent `java.lang.Object` class and overridden in the ADT.
Abstract Data Type (ADT)

Salient features of an ADT:

- Some entries (called constructors) have the same name as the class and no return type

- Some methods such as `equals()`, `hashCode()`, and `toString()` are inherited from the parent `java.lang.Object` class and overridden in the ADT
Abstract Data Type (ADT)

Salient features of an ADT:

- Some entries (called constructors) have the same name as the class and no return type
- Some entries (called methods) lack the `static` keyword and operate on data-type values
Abstract Data Type (ADT)

Salient features of an ADT:

- Some entries (called constructors) have the same name as the class and no return type
- Some entries (called methods) lack the `static` keyword and operate on data-type values
- Some methods such as `equals()`, `hashCode()`, and `toString()` are inherited from the parent `java.lang.Object` class and overridden in the ADT
Using an ADT

An object is an entity that can take on a data-type value.

Creating an object:

```java
<type> <name> = new <type>(<argument1>, <argument2>, ...);
```

Example:

```java
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```

<table>
<thead>
<tr>
<th>heads</th>
<th>tally</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;heads&quot;</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tails</th>
<th>tally</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;tails&quot;</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Using an ADT

An object is an entity that can take on a data-type value
An object is an entity that can take on a data-type value

Creating an object

```
<type> <name> = new <type>(<argument1>, <argument2>, ...);
```
Using an ADT

An object is an entity that can take on a data-type value

Creating an object

```
<type> <name> = new <type>(<argument1>, <argument2>, ...);
```

Example

```java
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```
Using an ADT

A method, invoked as 

\[
\text{<object>.	ext{name}(<argument1>, <argument2>, ...)}
\]

operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```

```
heads
"heads"
id 47
```

```
tally
```

```
tails
"tails"
id 53
```

```
StdOut.println(heads.tally());
StdOut.println(tails.tally());
```

```
47
53
```
A method, invoked as `object.<name>(argument1, argument2, ...)`, operates on data-type values.
Using an ADT

A method, invoked as `[object].<name>(<argument1>, <argument2>, ...)`, operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```
Using an ADT

A method, invoked as \texttt{[object].\textless name\textgreater (argument1, argument2, ...)}, operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```

```plaintext
<table>
<thead>
<tr>
<th>heads</th>
<th>tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>id</td>
</tr>
<tr>
<td>47</td>
<td>53</td>
</tr>
<tr>
<td>tally</td>
<td>tally</td>
</tr>
</tbody>
</table>
```

StdOut.println(heads.tally());
StdOut.println(tails.tally());
47
53
Using an ADT

A method, invoked as `<object>.<name>(<argument1>, <argument2>, ...), operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```

```
StdOut.println(heads.tally());
StdOut.println(tails.tally());
```
Using an ADT

A method, invoked as [object].<name>(<argument1>, <argument2>, ...), operates on data-type values

Example

```java
for (int i = 0; i < 100; i++) {
    if (StdRandom.bernoulli(0.5)) {
        heads.increment();
    } else {
        tails.increment();
    }
}
```

```
StdOut.println(heads.tally());
StdOut.println(tails.tally());
```

47
53
Using an ADT

Counter heads = new Counter("heads");
Counter tails = new Counter("tails");

Aliasing
Using an ADT

```scala
Counter heads = new Counter("heads");
Counter tails = new Counter("tails");
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>tally</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Using an ADT

Counter heads = new Counter("heads");
Counter tails = new Counter("tails");

Aliasing

heads = tails;
Using an ADT

Counter heads = new Counter("heads");
Counter tails = new Counter("tails");

Aliasing

heads = tails;
Using an ADT

Two objects \( x \) and \( y \) must be compared for equality as \( x.equals(y) \) and not as \( x == y \).

Example

```java
String x = " Hello , World ";
String y = " Hello , World ";
String z = " Cogito , ergo sum ";
StdOut.println("x == x? "+ (x == x));
StdOut.println("x == y? "+ (x == y));
StdOut.println("x == z? "+ (x == z));
StdOut.println("x. equals (x)? "+ x.equals(x));
StdOut.println("x. equals (y)? "+ x.equals(y));
StdOut.println("x. equals (z)? "+ x.equals(z));
```

```
x == x? true
x == y? false
x == z? false
x. equals (x)? true
x. equals (y)? true
x. equals (z)? false
```
Using an ADT

Two objects $x$ and $y$ must be compared for equality as $x.equals(y)$ and not as $x == y$. 

Example

```java
String x = " Hello , World ";
String y = " Hello , World ";
String z = " Cogito , ergo sum ";
StdOut.println("x == x? " + (x == x));
StdOut.println("x == y? " + (x == y));
StdOut.println("x == z? " + (x == z));
StdOut.println("x. equals(x)? " + x.equals(x));
StdOut.println("x. equals(y)? " + x.equals(y));
StdOut.println("x. equals(z)? " + x.equals(z));
```

```
x == x? true
x == y? false
x == z? false
x.equals(x)? true
x.equals(y)? true
x.equals(z)? false
```
Using an ADT

Two objects $x$ and $y$ must be compared for equality as $x.equals(y)$ and not as $x == y$

Example

```java
String x = "Hello, World";
String y = "Hello, World";
String z = "Cogito, ergo sum";
StdOut.println("x == x? " + (x == x));
StdOut.println("x == y? " + (x == y));
StdOut.println("x == z? " + (x == z));
StdOut.println("x.equals(x)? " + x.equals(x));
StdOut.println("x.equals(y)? " + x.equals(y));
StdOut.println("x.equals(z)? " + x.equals(z));
```
Using an ADT

Two objects \( x \) and \( y \) must be compared for equality as \( x.equals(y) \) and not as \( x == y \)

Example

```java
String x = "Hello, World";
String y = "Hello, World";
String z = "Cogito, ergo sum";
StdOut.println("x == x? " + (x == x));
StdOut.println("x == y? " + (x == y));
StdOut.println("x == z? " + (x == z));
StdOut.println("x.equals(x)? " + x.equals(x));
StdOut.println("x.equals(y)? " + x.equals(y));
StdOut.println("x.equals(z)? " + x.equals(z));
```

\[
x == x? true
x == y? false
x == z? false
x.equals(x)? true
x.equals(y)? true
x.equals(z)? false
\]
Using an ADT

Program: Flips.java

• Command-line input: n (int)
• Standard output: number of heads, tails, and the difference from n coin flips
Using an ADT

Program: Flips.java
Using an ADT

Program: Flips.java
- Command-line input: \( n \) (int)
Using an ADT

Program: Flips.java

- Command-line input: \( n \) (int)
- Standard output: number of heads, tails, and the difference from \( n \) coin flips
Using an ADT

Program: Flips.java

- Command-line input: n (int)
- Standard output: number of heads, tails, and the difference from n coin flips
Program: Flips.java

- Command-line input: \( n \) (int)
- Standard output: number of heads, tails, and the difference from \( n \) coin flips

```
> ~/workspace/dsaj/programs
$ java Flips 1000000
```
Using an ADT

Program: Flips.java

- Command-line input: \( n \) (int)
- Standard output: number of heads, tails, and the difference from \( n \) coin flips

```
> ~/workspace/dsaj/programs
$ java Flips 1000000
499771 Heads
500229 Tails
delta: 458
$ _
```
Using an ADT

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Flips {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        StdOut.println(heads);
        StdOut.println(tails);
        StdOut.println("delta: "+ Math.abs(heads.tally() - tails.tally()));
    }
}
```
Using an ADT

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Flips {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        StdOut.println(heads);
        StdOut.println(tails);
        StdOut.println("delta: " + Math.abs(heads.tally() - tails.tally()));
    }
}
```
Using an ADT

Program: FlipsMax.java
• Command-line input: n (int)
• Standard output: the winner from n coin flips
Using an ADT

Program: FlipsMax.java
Using an ADT

Program: FlipsMax.java

- Command-line input: n (int)
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips
Using an ADT

Program: FlipsMax.java

• Command-line input: $n$ (int)
• Standard output: the winner from $n$ coin flips

```bash
$ ~workspace/dsaj/programs
```
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```
$ java FlipsMax 1000000
```
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```bash
$ ~/workspace/dsaj/programs
$ java FlipsMax 1000000
500371 Heads wins
$ _
```
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```
$ ~/workspace/dsaj/programs
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
```
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```bash
$ ~/workspace/dsaj/programs
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ _
```
Using an ADT

Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```
> ~/workspace/dsaj/programs
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ java FlipsMax 1000000
```
Program: FlipsMax.java

- Command-line input: \( n \) (int)
- Standard output: the winner from \( n \) coin flips

```bash
~/workspace/dsaj/programs
$ java FlipsMax 1000000
500371 Heads wins
$ java FlipsMax 1000000
500776 Tails wins
$ java FlipsMax 1000000
500995 Tails wins
$ _
```
Using an ADT

```java
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class FlipsMax {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        if (heads.equals(tails)) {
            StdOut.println("Tie");
        } else {
            StdOut.println(max(heads, tails) + " wins");
        }
    }

    private static Counter max(Counter x, Counter y) {
        if (x.tally() > y.tally()) {
            return x;
        }
        return y;
    }
}
```
import java.util.Counter;
import java.io.StdOut;
import java.io.StdRandom;

public class FlipsMax {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Counter heads = new Counter("Heads");
        Counter tails = new Counter("Tails");
        for (int i = 0; i < n; i++) {
            if (StdRandom.bernoulli(0.5)) {
                heads.increment();
            } else {
                tails.increment();
            }
        }
        if (heads.equals(tails)) {
            StdOut.println("Tie");
        } else {
            StdOut.println(max(heads, tails) + " wins");
        }
    }

    private static Counter max(Counter x, Counter y) {
        if (x.tally() > y.tally()) {
            return x;
        }
        return y;
    }
}
Using an ADT

Program: Rolls.java

- Command-line input: n (int)
- Standard output: frequencies of face values from rolling n 6-sided dice
Using an ADT

Program: Rolls.java
Using an ADT

Program: Rolls.java

- Command-line input: $n$ (int)
Program: Rolls.java

- Command-line input: \( n \) (int)
- Standard output: frequencies of face values from rolling \( n \) 6-sided dice
Using an ADT

Program: Rolls.java

- Command-line input: \( n \) (int)
- Standard output: frequencies of face values from rolling \( n \) 6-sided dice

```bash
> ~/workspace/dsaj/programs

$ 
```
Using an ADT

Program: Rolls.java

- Command-line input: \( n \) (int)
- Standard output: frequencies of face values from rolling \( n \) 6-sided dice

```
$ java ~/workspace/dsaj/programs/Rolls 1000000
```
Using an ADT

Program: Rolls.java

- Command-line input: \( n \) (int)
- Standard output: frequencies of face values from rolling \( n \) 6-sided dice

```bash
$ ~/workspace/dsaj/programs
$ java Rolls 1000000
166923 1s
166543 2s
166528 3s
166373 4s
166517 5s
167116 6s
$ _
```
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Rolls {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int SIDES = 6;
        Counter[] rolls = new Counter[SIDES + 1];
        for (int i = 1; i <= SIDES; i++) {
            rolls[i] = new Counter(i + "s");
        }
        for (int j = 0; j < n; j++) {
            int result = StdRandom.uniform(1, SIDES + 1);
            rolls[result].increment();
        }
        for (int i = 1; i <= SIDES; i++) {
            StdOut.println(rolls[i]);
        }
    }
}
import dsa.Counter;
import stdlib.StdOut;
import stdlib.StdRandom;

public class Rolls {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int SIDES = 6;
        Counter[] rolls = new Counter[SIDES + 1];
        for (int i = 1; i <= SIDES; i++) {
            rolls[i] = new Counter(i + "s");
        }
        for (int j = 0; j < n; j++) {
            int result = StdRandom.uniform(1, SIDES + 1);
            rolls[result].increment();
        }
        for (int i = 1; i <= SIDES; i++) {
            StdOut.println(rolls[i]);
        }
    }
}
Examples of ADTs

In(String name) constructs an input stream from a file with the given name.

boolean isEmpty() returns true if this input stream is empty, and false otherwise.

double readDouble() reads and returns the next double from this input stream.

Out(String name) constructs an output stream from a file with the given name.

void println(Object x) prints an object and a newline to this output stream.

void print(Object x) prints an object to this output stream.

void printf(String fmt, Object... args) prints args to this output stream using format string fmt.
Examples of ADTs

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In(String name)</td>
<td>constructs an input stream from a file with the given name</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>returns true if this input stream is empty, and false otherwise</td>
</tr>
<tr>
<td>double readDouble()</td>
<td>reads and returns the next double from this input stream</td>
</tr>
</tbody>
</table>
## Examples of ADTs

### stdlib.In

- **In(String name)**: constructs an input stream from a file with the given name
- **boolean isEmpty()**: returns `true` if this input stream is empty, and `false` otherwise
- **double readDouble()**: reads and returns the next double from this input stream

### stdlib.Out

- **Out(String name)**: constructs an output stream from a file with the given name
- **void println(Object x)**: prints an object and a newline to this output stream
- **void print(Object x)**: prints an object to this output stream
- **void printf(String fmt, Object... args)**: prints `args` to this output stream using format string `fmt`
Examples of ADTs

Java.lang.String

String()
creates an empty string

int length()
returns the length of the string

char charAt(int i)
returns the character in the string at index i

String substring(int i, int j)
returns a substring of the string from index i (inclusive) to index j (exclusive)

Example

public static boolean isPalindrome (String s) {
    int n = s.length();
    if (n < 2) {
        return true;
    }
    return s.charAt(0) == s.charAt(n - 1) && isPalindrome(s.substring(1, n - 1));
}
**Examples of ADTs**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String()</code></td>
<td>creates an empty string</td>
</tr>
<tr>
<td><code>int length()</code></td>
<td>returns the length of the string</td>
</tr>
<tr>
<td><code>char charAt(int i)</code></td>
<td>returns the character in the string at index i</td>
</tr>
<tr>
<td><code>String substring(int i, int j)</code></td>
<td>returns a substring of the string from index i (inclusive) to index j (exclusive)</td>
</tr>
</tbody>
</table>
Examples of ADTs

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String()</td>
<td>creates an empty string</td>
</tr>
<tr>
<td>int length()</td>
<td>returns the length of the string</td>
</tr>
<tr>
<td>char charAt(int i)</td>
<td>returns the character in the string at index i</td>
</tr>
<tr>
<td>String substring(int i, int j)</td>
<td>returns a substring of the string from index i (inclusive) to index j (exclusive)</td>
</tr>
</tbody>
</table>

Example

```java
public static boolean isPalindrome(String s) {
    int n = s.length();
    if (n < 2) {
        return true;
    }
    return s.charAt(0) == s.charAt(n - 1) && isPalindrome(s.substring(1, n - 1));
}
```
Examples of ADTs
Examples of ADTs
Examples of ADTs

```
    0  1  2  3  4
   ●  ●  ●  ●  ●

    5  6  7  8  9
   ●  ●  ●  ●  ●

   4  3
```
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs
Examples of ADTs

- WeightedQuickUnionUF implements dsa.UF

  - WeightedQuickUnionUF(int n) constructs an empty union-find data structure with \( n \) sites.
  - int find(int p) returns the canonical site of the component containing site \( p \).
  - int count() returns the number of components.
  - boolean connected(int p, int q) returns true if sites \( p \) and \( q \) belong to the same component, and false otherwise.
  - void union(int p, int q) connects sites \( p \) and \( q \).
Examples of ADTs

dsa.WeightedQuickUnionUF implements dsa.UF

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WeightedQuickUnionUF(int n)</td>
<td>constructs an empty union-find data structure with n sites</td>
</tr>
<tr>
<td>int find(int p)</td>
<td>returns the canonical site of the component containing site p</td>
</tr>
<tr>
<td>int count()</td>
<td>returns the number of components</td>
</tr>
<tr>
<td>boolean connected(int p, int q)</td>
<td>returns true if sites p and q belong to the same component, and false otherwise</td>
</tr>
<tr>
<td>void union(int p, int q)</td>
<td>connects sites p and q</td>
</tr>
</tbody>
</table>
Examples of ADTs

Program:

Components.java

• Standard input: n (int) and a sequence of pairs of integers representing sites

• Standard output: number of components left after merging the sites that are in different components
Examples of ADTs

Program: Components.java
Examples of ADTs

Program: Components.java

- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites

Standard output: number of components left after merging the sites that are in different components
Examples of ADTs

Program: Components.java

- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components
Examples of ADTs

Program: Components.java
- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components
Examples of ADTs

Program: Components.java

- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

```
$ cat ../data/tinyUF.txt
```
Examples of ADTs

Program: Components.java

- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

```bash
> ~/workspace/dsaj/programs
$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ _
```
Examples of ADTs

Program: Components.java

- Standard input: $n$ (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

```
$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ java Components < ../data/tinyUF.txt
```
Examples of ADTs

Program: Components.java

- Standard input: \( n \) (int) and a sequence of pairs of integers representing sites
- Standard output: number of components left after merging the sites that are in different components

```
$ cat ../data/tinyUF.txt
10
4 3
3 8
6 5
9 4
2 1
8 9
5 0
7 2
6 1
1 0
6 7
$ java Components < ../data/tinyUF.txt
2 components
$ _
```
Examples of ADTs

```
import dsa.WeightQuickUnionUF;
import stdlib.StdIn;
import stdlib.StdOut;

public class Components{
    public static void main(String[] args){
        int n = StdIn.readInt();
        WeightedQuickUnionUF uf = new WeightedQuickUnionUF(n);
        while(!StdIn.isEmpty()){
            int p = StdIn.readInt();
            int q = StdIn.readInt();
            uf.union(p, q);
        }
        StdOut.println(uf.count() + " components ");
    }
}
```
import dsa.WeightedQuickUnionUF;
import stdlib.StdIn;
import stdlib.StdOut;

public class Components {
    public static void main(String[] args) {
        int n = StdIn.readInt();
        WeightedQuickUnionUF uf = new WeightedQuickUnionUF(n);
        while (!StdIn.isEmpty()) {
            int p = StdIn.readInt();
            int q = StdIn.readInt();
            uf.union(p, q);
        }
        StdOut.println(uf.count() + " components");
    }
}
Defining an ADT
Defining an ADT

```java
package dsa;

// Import statements.
...

// Class definition.
public class Program [implements <name>] {
    // Field declarations.
    ...

    // Constructor definitions.
    ...

    // Method definitions.
    ...

    // Function definitions.
    ...

    // Inner class definitions.
    ...
}
```
Defining an ADT

Field declaration statement
private | public [ static ] <type> <name>;

Fields are accessed as [<target>.]<name>, where <target> is an object name for an instance field and a library name for a static field.

Example:
• Instance fields String id and int count in Counter
• Static field double PI in Math
Defining an ADT

Field declaration statement

```java
private|public [static] <type> <name>;
```
Defining an ADT

Field declaration statement

```java
private/public [static] <type> <name>;
```

Fields are accessed as `<target>.<name>`, where `<target>` is an object name for an instance field and a library name for a `static` field.
Defining an ADT

Field declaration statement

```
private | public [static] <type> <name>;
```

Fields are accessed as `[<target>]<name>`, where `<target>` is an object name for an instance field and a library name for a static field.

Example:
Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `target.<name>`, where `target` is an object name for an instance field and a library name for a static field.

Example:

- **Instance fields** `String id` and `int count` in `Counter`
Defining an ADT

Field declaration statement

```
private|public [static] <type> <name>;
```

Fields are accessed as `<target>.[<name>]`, where `<target>` is an object name for an instance field and a library name for a static field.

Example:

- Instance fields `String id` and `int count` in `Counter`
- Static field `double PI` in `Math`
Defining an ADT

Constructor definition

private | public <name > ( < parameter1 >, <parameter2 >, ...) {
<statement >
...
}

where <name > is the name of the ADT

Example (Counter.java)

public Counter ( String id) {
this .id = id;
count = 0;
}
Defining an ADT

Constructor definition

```java
private|public <name>((<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}

where <name> is the name of the ADT
```
Defining an ADT

Constructor definition

```java
private | public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example (Counter.java)

```java
public Counter(String id) {
    this.id = id;
    count = 0;
}
```
Defining an ADT

Constructor definition

```java
private | public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example (`Counter.java`)

```java
public Counter(String id) {
    this.id = id;
    count = 0;
}
```

Within a constructor, `this` is a reference to the object being constructed
Defining an ADT

Constructor definition

```java
private|public <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

where `<name>` is the name of the ADT

Example (`Counter.java`)

```java
public Counter(String id) {
    this.id = id;
    count = 0;
}
```

Within a constructor, `this` is a reference to the object being constructed

If an ADT has no explicit constructors, `javac` implicitly provides an empty constructor
Defining an ADT

Method definition

private | public void |<type > <name >( < parameter1 >, <parameter2 >, ...) {
<statement >
...
}

Example (Counter.java)

public void increment (){
count ++;
}

count;

public int tally () {
return count;
}
Defining an ADT

Method definition

```java
private | public void <type> <name> (<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Example (`Counter.java`)

```java
public void increment () {
    count ++;
}
```

```java
public int tally () {
    return count;
}
```

Within a method, `this` is a reference to the object on which the method was invoked.
Method definition

```java
private | public void <type> <name>(<parameter1>, <parameter2>, ...) {
   <statement>
   ...
}
```

Example (Counter.java)

```java
public void increment() {
   count++;
}

public int tally() {
   return count;
}
```
Method definition

```java
private | public void |<type> <name>(<parameter1>, <parameter2>, ...) {
    <statement>
    ...
}
```

Example (Counter.java)

```java
public void increment() {
    count++;
}

public int tally() {
    return count;
}
```

Within a method, *this* is a reference to the object on which the method was invoked
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's " + elephant.sound() + "!");
StdOut.println("A tiger's " + tiger.sound() + "!");
```

An elephant's trumpet!
A tiger's roar!
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

```
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's " + elephant.sound() + "!");
StdOut.println("A tiger's " + tiger.sound() + "!");

An elephant's trumpet!
A tiger's roar!
```
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's "+elephant.sound()+"!");
StdOut.println("A tiger's "+tiger.sound()+"!");
```
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

Example

```java
public interface Animal {
    public String sound();
}
```

```java
public class Elephant implements Animal {
    public String sound () {
        return " trumpet ";
    }
}
```

```java
public class Tiger implements Animal {
    public String sound () {
        return " roar ";
    }
}
```

```java
Animal elephant = new Elephant ();
Animal tiger = new Tiger ();
StdOut . println ("An elephant 's " + elephant . sound () + "!");
StdOut . println ("A tiger 's " + tiger . sound () + "!");
```

An elephant 's trumpet !
A tiger 's roar !
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's "+elephant.sound()+"!");
StdOut.println("A tiger's "+tiger.sound()+"!");

An elephant's trumpet!
A tiger's roar!
```
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API.

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's " + elephant.sound() + "!");
StdOut.println("A tiger's " + tiger.sound() + "!");
```

An elephant's trumpet!
A tiger's roar!
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant’s " + elephant.sound() + "!");
StdOut.println("A tiger’s " + tiger.sound() + "!");
```
Defining an ADT

An interface provides a formal mechanism for describing an ADT's API and supporting different implementations of that API

Example

```java
public interface Animal {
    public String sound();
}

public class Elephant implements Animal {
    public String sound() {
        return "trumpet";
    }
}

public class Tiger implements Animal {
    public String sound() {
        return "roar";
    }
}

Animal elephant = new Elephant();
Animal tiger = new Tiger();
StdOut.println("An elephant's " + elephant.sound() + "!");
StdOut.println("A tiger's " + tiger.sound() + "!");

An elephant's trumpet!
A tiger's roar!
```
Comparing interfaces

**java.lang.Comparable**

```java
int compareTo(Type other) returns a comparison of this object with other
```

**java.util.Comparator**

```java
int compare(Type v, Type w) returns a comparison of object v with object w
```
## Defining an ADT

### Comparison interfaces

#### java.lang.Comparable

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int compareTo(Type other)</td>
<td>returns a comparison of this object with other</td>
</tr>
</tbody>
</table>

#### java.util.Comparator

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int compare(Type v, Type w)</td>
<td>returns a comparison of object v with object w</td>
</tr>
</tbody>
</table>
Defining an ADT

ComparableADT.java

```java
import java.util.Comparator;

public class ComparableADT implements Comparable<ComparableADT> {
    // Natural ordering.
    public int compareTo(ComparableADT other) {
        ...
    }
    public static Comparator<ComparableADT> aOrder() {
        return new AOrder();
    }
    public static Comparator<ComparableADT> bOrder() {
        return new BOrder();
    }
    // Alternate ordering 1.
    private static class AOrder implements Comparator<ComparableADT> {
        ...
        public int compare(ComparableADT v, ComparableADT w) {
            ...
        }
    }
    // Alternate ordering 2.
    private static class BOrder implements Comparator<ComparableADT> {
        ...
        public int compare(ComparableADT v, ComparableADT w) {
            ...
        }
    }
    ...
}
```
```java
import java.util.Comparator;

public class ComparableADT implements Comparable<ComparableADT> {
    // Natural ordering.
    public int compareTo(ComparableADT other) {
        ...
    }

    public static Comparator<ComparableADT> aOrder() {
        return new AOrder();
    }

    public static Comparator<ComparableADT> bOrder() {
        return new BOrder();
    }

    // Alternate ordering 1.
    private static class AOrder implements Comparator<ComparableADT> {
        ...
        public int compare(ComparableADT v, ComparableADT w) {
            ...
        }
    }

    // Alternate ordering 2.
    private static class BOrder implements Comparator<ComparableADT> {
        ...
        public int compare(ComparableADT v, ComparableADT w) {
            ...
        }
    }
    ...
}
```
Defining an ADT

list
dsa.Counter implements java.lang.Comparable<Counter>

Counter(String id) constructs a counter given its id

void increment() increments this counter by 1

int tally() returns the current value of this counter

void reset() resets this counter to zero

boolean equals(Object other) returns true if this counter and other have the same tally, and false otherwise

String toString() returns a string representation of this counter

int compareTo(Counter other) returns a comparison of this counter with other by their tally
Defining an ADT

dsa.Counter implements java.lang.Comparable<Counter>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor Counter(String id)</td>
<td>constructs a counter given its id</td>
</tr>
<tr>
<td>void increment()</td>
<td>increments this counter by 1</td>
</tr>
<tr>
<td>int tally()</td>
<td>returns the current value of this counter</td>
</tr>
<tr>
<td>void reset()</td>
<td>resets this counter to zero</td>
</tr>
<tr>
<td>boolean equals(Object other)</td>
<td>returns true if this counter and other have the same tally, and false otherwise</td>
</tr>
<tr>
<td>String toString()</td>
<td>returns a string representation of this counter</td>
</tr>
<tr>
<td>int compareTo(Counter other)</td>
<td>returns a comparison of this counter with other by their tally</td>
</tr>
</tbody>
</table>
Defining an ADT

Program: Counter.java

- Command-line input: n (int), trials (int)
- Standard output: frequencies obtained from trials random draws of numbers from the interval [0, n)
Defining an ADT

Program: Counter.java

Command-line input: $n\,$(int), $trials\,$(int)

Standard output: frequencies obtained from $trials$ random draws of numbers from the interval $[0, n]$.
Defining an ADT

Program: Counter.java

- Command-line input: \( n \) (int), \( trials \) (int)
Program: Counter.java

- Command-line input: $n$ (int), $trials$ (int)
- Standard output: frequencies obtained from $trials$ random draws of numbers from the interval $[0, n)$
Defining an ADT

Program: Counter.java

- Command-line input: \( n \) (int), \( trials \) (int)
- Standard output: frequencies obtained from \( trials \) random draws of numbers from the interval \([0, n)\)
Program: Counter.java

- Command-line input: \( n \) (int), \( trials \) (int)
- Standard output: frequencies obtained from \( trials \) random draws of numbers from the interval \([0, n)\)

```bash
> ~/workspace/dsaj/programs
$ java dsa.Counter 2 1000
```
Defining an ADT

Program: Counter.java

• Command-line input: $n$ (int), $trials$ (int)
• Standard output: frequencies obtained from $trials$ random draws of numbers from the interval $[0, n)$

```
> ~/workspace/dsaj/programs

$ java dsa.Counter 2 1000
501 counter 0
499 counter 1
$ 
```
Defining an ADT

Counter.java

```java
package dsa;
import stdlib.StdOut;
import stdlib.StdRandom;
public class Counter implements Comparable<Counter> {
    private String id;
    private int count;
    public Counter(String id) {
        this.id = id;
        count = 0;
    }
    public void increment() {
        count++;
    }
    public int tally() {
        return count;
    }
    public void reset() {
        count = 0;
    }
    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other == this) {
            return true;
        }
        if (other.getClass() != this.getClass()) {
            return false;
        }
        // Equality check...
    }
    // Additional methods...
}
```
package dsa;

import stdlib.StdOut;
import stdlib.StdRandom;

public class Counter implements Comparable<Counter> {
    private String id;
    private int count;

    public Counter(String id) {
        this.id = id;
        count = 0;
    }

    public void increment() {
        count ++;
    }

    public int tally() {
        return count;
    }

    public void reset() {
        count = 0;
    }

    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other == this) {
            return true;
        }
        if (other.getClass() != this.getClass()) {
            return false;
        }
        if (other.equals(id)) {
            return false;
        }
        if (other.equals(count)) {
            return false;
        }
        return true;
    }
}

Defining an ADT
Defining an ADT

Counter.java

```java
Counter a = this, b = (Counter) other;
return a.count == b.count;
}

public String toString() {
    return count + " " + id;
}

public int compareTo(Counter other) {
    return this.count - other.count;
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    int trials = Integer.parseInt(args[1]);
    Counter[] hits = new Counter[n];
    for (int i = 0; i < n; i++) {
        hits[i] = new Counter("counter " + i);
    }
    for (int t = 0; t < trials; t++) {
        hits[StdRandom.uniform(n)].increment();
    }
    for (int i = 0; i < n; i++) {
        StdOut.println(hits[i]);
    }
}
```
Defining an ADT

list

dsa.Transaction implements java.lang.Comparable<Transaction>

Transaction(String name, Date date, double amount) constructs a transaction from a name, date, and amount.

Transaction(String s) constructs a transaction from a string s of the form "name date amount".

String name() returns the name of the person involved in this transaction.

Date date() returns the date of this transaction.

double amount() returns the amount of this transaction.

int hashCode() returns a hash code for this transaction.

String toString() returns a string representation of this transaction.

int compareTo(Transaction other) returns a comparison of this transaction with other by amount.

static Comparator<Transaction> nameOrder() returns a comparator for comparing two transactions by name.

static Comparator<Transaction> dateOrder() returns a comparator for comparing two transactions by date.
### Defining an ADT

**dsa.Transaction implements java.lang.Comparable<Transaction>**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction(String name, Date date, double amount)</td>
<td>constructs a transaction from a name, date, and amount</td>
</tr>
<tr>
<td>Transaction(String s)</td>
<td>constructs a transaction from a string s of the form “name date amount”</td>
</tr>
<tr>
<td>String name()</td>
<td>returns the name of the person involved in this transaction</td>
</tr>
<tr>
<td>Date date()</td>
<td>returns the date of this transaction</td>
</tr>
<tr>
<td>double amount()</td>
<td>returns the amount of this transaction</td>
</tr>
<tr>
<td>int hashCode()</td>
<td>returns a hash code for this transaction</td>
</tr>
<tr>
<td>String toString()</td>
<td>returns a string representation of this transaction</td>
</tr>
<tr>
<td>int compareTo(Transaction other)</td>
<td>returns a comparison of this transaction with other by amount</td>
</tr>
<tr>
<td>static Comparator&lt;Transaction&gt; nameOrder()</td>
<td>returns a comparator for comparing two transactions by name</td>
</tr>
<tr>
<td>static Comparator&lt;Transaction&gt; dateOrder()</td>
<td>returns a comparator for comparing two transactions by date</td>
</tr>
</tbody>
</table>
Defining an ADT

Program: Transaction.java

• Standard output: four transactions (one per line) in different orders
Defining an ADT

Program: Transaction.java
Defining an ADT

Program: Transaction.java

- Standard output: four transactions (one per line) in different orders
Program: Transaction.java

- Standard output: four transactions (one per line) in different orders
Defining an ADT

Program: Transaction.java

- Standard output: four transactions (one per line) in different orders

```

> ~/workspace/dsaj/programs

$ java dsa.Transaction
```
## Defining an ADT

**Program:** Transaction.java

- Standard output: four transactions (one per line) in different orders

```
> ~/workspace/dsaj/programs

$ java dsa.Transaction

Unsorted:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td>6/17/1990</td>
<td>644.08</td>
</tr>
<tr>
<td>Tarjan</td>
<td>3/26/2002</td>
<td>4121.85</td>
</tr>
<tr>
<td>Knuth</td>
<td>6/14/1999</td>
<td>288.34</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>8/22/2007</td>
<td>2678.40</td>
</tr>
</tbody>
</table>

Sorted by name:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra</td>
<td>8/22/2007</td>
<td>2678.40</td>
</tr>
<tr>
<td>Knuth</td>
<td>6/14/1999</td>
<td>288.34</td>
</tr>
<tr>
<td>Tarjan</td>
<td>3/26/2002</td>
<td>4121.85</td>
</tr>
<tr>
<td>Turing</td>
<td>6/17/1990</td>
<td>644.08</td>
</tr>
</tbody>
</table>

Sorted by date:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turing</td>
<td>6/17/1990</td>
<td>644.08</td>
</tr>
<tr>
<td>Knuth</td>
<td>6/14/1999</td>
<td>288.34</td>
</tr>
<tr>
<td>Tarjan</td>
<td>3/26/2002</td>
<td>4121.85</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>8/22/2007</td>
<td>2678.40</td>
</tr>
</tbody>
</table>

Sorted by amount:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knuth</td>
<td>6/14/1999</td>
<td>288.34</td>
</tr>
<tr>
<td>Turing</td>
<td>6/17/1990</td>
<td>644.08</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>8/22/2007</td>
<td>2678.40</td>
</tr>
<tr>
<td>Tarjan</td>
<td>3/26/2002</td>
<td>4121.85</td>
</tr>
</tbody>
</table>
```

$ _
Defining an ADT

package dsa;
import java.util.Arrays;
import java.util.Comparator;
import stdlib.StdOut;

public class Transaction implements Comparable<Transaction> {
    private String name;
    private Date date;
    private double amount;

    public Transaction(String name, Date date, double amount) {
        this.name = name;
        this.date = date;
        this.amount = amount;
    }

    public Transaction(String s) {
        String[] a = s.split(\s+);
        name = a[0];
        date = new Date(a[1]);
        amount = Double.parseDouble(a[2]);
    }

    public String name() {
        return name;
    }

    public Date date() {
        return date;
    }

    public double amount() {
        return amount;
    }

    @Override
    public int compareTo(Transaction o) {
        return name.compareTo(o.name);
    }
}


package dsa;

import java.util.Arrays;
import java.util.Comparator;
import stdlib.StdOut;

public class Transaction implements Comparable<Transaction> {
    private String name;
    private Date date;
    private double amount;

    public Transaction(String name, Date date, double amount) {
        this.name = name;
        this.date = date;
        this.amount = amount;
    }

    public Transaction(String s) {
        String[] a = s.split("\s+");
        name = a[0];
        date = new Date(a[1]);
        amount = Double.parseDouble(a[2]);
    }

    public String name() {
        return name;
    }

    public Date date() {
        return date;
    }

    public double amount() {
        return amount;
    }
}

public int hashCode() {
    int hash = 1;
    hash = 31 * hash + name.hashCode();
    hash = 31 * hash + date.hashCode();
    hash = 31 * hash + ((Double) amount).hashCode();
    return hash;
}

public String toString() {
    return String.format("%10s %10s %8.2f", name, date, amount);
}

public int compareTo(Transaction other) {
    return Double.compare(this.amount, other.amount);
}

public static Comparator<Transaction> nameOrder() {
    return new NameOrder();
}

public static Comparator<Transaction> dateOrder() {
    return new DateOrder();
}

private static class NameOrder implements Comparator<Transaction> {
    public int compare(Transaction v, Transaction w) {
        return v.name.compareTo(w.name);
    }
}

private static class DateOrder implements Comparator<Transaction> {
    public int compare(Transaction v, Transaction w) {
        return v.date.compareTo(w.date);
    }
}
public static void main(String[] args) {
    Transaction[] transactions = new Transaction[4];
    transactions[0] = new Transaction("Turing 6/17/1990 644.08");
    StdOut.println("Unsorted:");
    for (int i = 0; i < transactions.length; i++) {
        StdOut.println(transactions[i]);
    }
    StdOut.println();
    StdOut.println("Sorted by name:");
    Arrays.sort(transactions, Transaction.nameOrder());
    for (int i = 0; i < transactions.length; i++) {
        StdOut.println(transactions[i]);
    }
    StdOut.println();
    StdOut.println("Sorted by date:");
    Arrays.sort(transactions, Transaction.dateOrder());
    for (int i = 0; i < transactions.length; i++) {
        StdOut.println(transactions[i]);
    }
    StdOut.println();
    StdOut.println("Sorted by amount:");
    Arrays.sort(transactions);
    for (int i = 0; i < transactions.length; i++) {
        StdOut.println(transactions[i]);
    }
}
Defining an ADT

Iteration interfaces

/list
/java.lang.Iterable
/Iterator<Type>

iterator() returns an iterator over a collection of items of type Type

/list
/java.util.Iterator

hasNext() returns true if the iterator has more items, and false otherwise

next() returns the next item in the iterator
### Defining an ADT

#### Iteration interfaces

<table>
<thead>
<tr>
<th>java.lang.Iterable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iterator&lt;Type&gt; iterator()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>java.util.Iterator</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean hasNext()</td>
</tr>
<tr>
<td>Type next()</td>
</tr>
</tbody>
</table>
Defining an ADT

An Iterable object $o$ can be iterated over using the for-each statement

```java
for ( Type item : o) {
    <statement>
    ...
}
```

which is equivalent to

```java
Iterator iter = o.iterator();
while ( iter.hasNext()) {
    Type item = iter.next();
    <statement>
    ...
}
```

Arrays are iterable, and thus can be iterated using the for-each statement

Example

```java
String[] dow = {" Sun ", " Mon ", " Tue ", " Wed ", " Thu ", " Fri ", " Sat "};
for ( String s : dow ) {
    StdOut.println(s);
}
```
An **Iterable** object `o` can be iterated over using the for-each statement

```java
for (Type item : o) {
    <statement>
    ...
}
```

which is equivalent to

```java
Iterator iter = o.iterator();
while (iter.hasNext()) {
    Type item = iter.next();
    <statement>
    ...
}
```
An `Iterable` object `o` can be iterated over using the for-each statement

```java
for (Type item : o) {
    <statement>
    ...
}
```

which is equivalent to

```java
Iterator iter = o.iterator();
while (iter.hasNext()) {
    Type item = iter.next();
    <statement>
    ...
}
```

Arrays are iterable, and thus can be iterated using the for-each statement
An iterable object `o` can be iterated over using the for-each statement

```java
for (Type item : o) {
    <statement>
    ...
}
```

which is equivalent to

```java
Iterator iter = o.iterator();
while (iter.hasNext()) {
    Type item = iter.next();
    <statement>
    ...
}
```

Arrays are iterable, and thus can be iterated using the for-each statement

Example

```java
String[] dow = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"};
for (String s : dow) {
    StdOut.println(s);
}
```
Defining an ADT

```java
import java.util.Iterator;

public class IterableADT implements Iterable<Type> {
    ...
    public Iterator<Type> iterator() {
        return new AnIterator();
    }
    private class AnIterator implements Iterator<Type> {
        ...
        public boolean hasNext() {
            ...
        }
        public Type next() {
            ...
        }
    }
    ...
}
```
Defining an ADT

```java
import java.util.Iterator;

public class IterableADT implements Iterable<Type> {
    ...
    public Iterator<Type> iterator() {
        return new AnIterator();
    }
    ...
    private class AnIterator implements Iterator<Type> {
        ...
        public boolean hasNext() {
            ...
        }
        public Type next() {
            ...
        }
    }
    ...
}
Defining an ADT

Command-line input: sentence (String)
Standard output: the words in sentence, one per line
Defining an ADT

Program:  Words.java
Defining an ADT

Program: Words.java

• Command-line input: sentence (String)
Defining an ADT

Program:  Words.java

- Command-line input:  sentence (String)
- Standard output:  the words in sentence, one per line
Defining an ADT

Program: Words.java

- Command-line input: sentence (String)
- Standard output: the words in sentence, one per line
Defining an ADT

Program: Words.java

• Command-line input: sentence (String)
• Standard output: the words in sentence, one per line

```bash
~/workspace/dsaj/programs
$ java Words "it was the best of times it was the worst of times"
```

Defining an ADT

Program: Words.java

- Command-line input: `sentence` (String)
- Standard output: the words in `sentence`, one per line

```
> ~/workspace/dsaj/programs

$ java Words "it was the best of times it was the worst of times"
it
was
the
best
of
times
it
was
the
worst
of
times
$ _
```
Defining an ADT

Words.java

```java
import java.util.Iterator;
import stdlib.StdOut;

public class Words implements Iterable<String> {
    private String sentence;

    public Words(String sentence) {
        this.sentence = sentence;
    }

    public Iterator<String> iterator() {
        return new WordsIterator();
    }

    private class WordsIterator implements Iterator<String> {
        private String[] words;
        private int i;

        public WordsIterator() {
            words = sentence.split("\s+"塌独);  
            i = 0;
        }

        public boolean hasNext() {
            return i < words.length;
        }

        public String next() {
            return words[i++];
        }
    }
}
```

public static void main(String[] args) {
    String sentence = args[0];
}
Defining an ADT

```java
import java.util.Iterator;
import stdlib.StdOut;

class Words implements Iterable<String> {
    private String sentence;

    public Words(String sentence) {
        this.sentence = sentence;
    }

    public Iterator<String> iterator() {
        return new WordsIterator();
    }

    private class WordsIterator implements Iterator<String> {
        private String[] words;
        private int i;

        public WordsIterator() {
            words = sentence.split("\s+" );
            i = 0;
        }

        public boolean hasNext() {
            return i < words.length;
        }

        public String next() {
            return words[i++];
        }
    }
}

public static void main(String[] args) {
    String sentence = args[0];
    StdOut.println(sentence);
}
```
Words words = new Words(sentence);
for (String word : words) {
    StdOut.println(word);
}
Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running

Example:
ArrayIndexOutOfBoundsException and NullPointerException

Throwing an exception

throw new <exception> ( <message> );

Example

throw new IllegalArgumentException ("x must be positive");
Errors (aka exceptions) are disruptive events that occur while a program is running.

Example:

```java
throw new ArrayIndexOutOfBoundsException();
```

Example:

```java
throw new NullPointerException();
```

Example:

```java
throw new IllegalArgumentException("x must be positive");
```
Errors (aka exceptions) are disruptive events that occur while a program is running.

Example: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Example: `throw new IllegalArgumentException(“x must be positive”);`
Error Handling

Errors (aka exceptions) are disruptive events that occur while a program is running.

Example: `ArrayIndexOutOfBoundsException` and `NullPointerException`

Throwing an exception

```java
throw new <exception>(<message>);
```
Errors (aka exceptions) are disruptive events that occur while a program is running.

**Example:** ArrayIndexOutOfBoundsException and NullPointerException

**Throwing an exception**

```java
throw new <exception>(<message>);
```

**Example**

```java
throw new IllegalArgumentException("x must be positive");
```
Error Handling

Catching an exception

```java
try {
    <statement>
    ...
} catch (< exception > e) {
    <statement>
    ...
} catch (< exception > e) {
    <statement>
    ...
}...
finally {
    <statement>
    ...
}
```
Catching an exception

```java
try {
    <statement>
    ...
}
catch (<exception> e) {
    <statement>
    ...
}
catch (<exception> e) {
    <statement>
    ...
}
catch (<exception> e) {
    <statement>
    ...
}
finally {
    <statement>
    ...
}
...
Error Handling

Program: ErrorHandling.java

• Command-line input: \( x \) (double)

• Standard output: the square root of \( x \)
Error Handling

Program: ErrorHandling.java
Program: ErrorHandling.java

- Command-line input: \( x \) (double)
Program: ErrorHandling.java

- Command-line input: $x$ (double)
- Standard output: the square root of $x$
Error Handling

Program: ErrorHandling.java

- Command-line input: x (double)
- Standard output: the square root of x

```bash
> ~/workspace/dsaj/programs

$ _
```
Program:  ErrorHandling.java

- Command-line input:  \( x \) (double)
- Standard output: the square root of \( x \)
Program: ErrorHandling.java

- Command-line input: $x$ (double)
- Standard output: the square root of $x$

```bash
$ ~/workspace/dsaj/programs
$ java ErrorHandling
x not specified
Done!
$ _
```
Program: ErrorHandling.java

- Command-line input: $x$ (double)
- Standard output: the square root of $x$

```bash
$ ~/workspace/dsaj/programs
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
```
Error Handling

Program:  ErrorHandling.java

- Command-line input:  \( x \) (double)
- Standard output:  the square root of \( x \)

```
$ ~/workspace/dsaj/programs
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ _
```
Program: ErrorHandling.java

- Command-line input: \( x \) (double)
- Standard output: the square root of \( x \)

```
> ~/workspace/dsaj/programs
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
```
Program: ErrorHandling.java

- Command-line input: \( x \) (double)
- Standard output: the square root of \( x \)

```
$ ~/workspace/dsaj/programs
$ java ErrorHandling
  x not specified
  Done!
$ java ErrorHandling two
  x must be a double
  Done!
$ java ErrorHandling -2
  x must be positive
  Done!
$ _
```
**Error Handling**

**Program:** ErrorHandling.java

- **Command-line input:** $x$ (double)
- **Standard output:** the square root of $x$

```
$ ~/workspace/dsaj/programs
$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
x must be positive
Done!
$ java ErrorHandling 2
```
Program:  ErrorHandling.java

- Command-line input:  \( x \) (double)
- Standard output:  the square root of \( x \)

```
>  ~/workspace/dsaj/programs

$ java ErrorHandling
x not specified
Done!
$ java ErrorHandling two
x must be a double
Done!
$ java ErrorHandling -2
x must be positive
Done!
$ java ErrorHandling 2
1.4142135623730951
Done!
$ _
```
import stdlib.StdOut;

public class ErrorHandling {
    public static void main(String[] args) {
        try {
            double x = Double.parseDouble(args[0]);
            double result = Math.sqrt(x);
            StdOut.println(result);
        } catch (ArrayIndexOutOfBoundsException e) {
            StdOut.println("x not specified");
        } catch (NumberFormatException e) {
            StdOut.println("x must be a double");
        } catch (IllegalArgumentException e) {
            StdOut.println(e.getMessage());
        } finally {
            StdOut.println("Done!");
        }
    }

    private static double sqrt(double x) {
        if (x < 0) {
            throw new IllegalArgumentException("x must be positive");
        }
        return Math.sqrt(x);
    }
}
import stdlib.StdOut;

public class ErrorHandling {
    public static void main(String[] args) {
        try {
            double x = Double.parseDouble(args[0]);
            double result = sqrt(x);
            StdOut.println(result);
        } catch (ArrayIndexOutOfBoundsException e) {
            StdOut.println("x not specified");
        } catch (NumberFormatException e) {
            StdOut.println("x must be a double");
        } catch (IllegalArgumentException e) {
            StdOut.println(e.getMessage());
        } finally {
            StdOut.println("Done!");
        }
    }

    private static double sqrt(double x) {
        if (x < 0) {
            throw new IllegalArgumentException("x must be positive");
        }
        return Math.sqrt(x);
    }
}