Homework Assignment #3
Please take a look at the Simple and Fast Multimedia Library and install it on your system:

http://www.sfml-dev.org/

We will use this library for our final homework programming task. I will present the task on Thursday.

Modeling

**Abstraction** is the classification of phenomena into concepts.

**Modeling** is the development of abstractions that can be used to answer specific questions about a set of phenomena.

Modeling is what you do as a software engineer when designing a software system.

You abstract concepts from the **application domain** (the system’s environment) and from the **solution domain** (technologies) to derive a simple model that is easy to understand.

UML Basics

UML is a standard modeling language, particularly for object-oriented modeling.

For system development, we use UML to represent three different models of the system:

- the **functional model**, represented with **use case diagrams** (system from the user’s perspective),
- the **object model**, represented with **class diagrams** (structure of the system),
- the **dynamic model**, represented with **sequence diagrams**, **statechart diagrams**, and **activity diagrams** (internal behavior of the system).

Use Case Diagrams

Use case diagrams represent the functionality of the system.

A **use case** is a function of the system that yields a visible result for an actor.

An **actor** is any person or object outside the system that interacts with the system.

In use case diagrams, the actors are outside the boundary of the system, and the use cases are inside.

We indicate use cases with **ovals** (ellipses).

Example:

Use case diagram for a simple watch.

Actors

An actor models an external entity which communicates with the system:

- User
- External system
- Physical environment

An actor has a unique name and an optional description.

Examples:

- Passenger: A person in the train
- GPS satellite: Provides the system with GPS coordinates
Use Cases
A use case represents a class of functionality provided by the system as an event flow.
A use case consists of:
• Unique name
• Participating actors
• Entry conditions
• Flow of events
• Exit conditions
• Special requirements

Use Cases: Example
Name: Purchase ticket
Participating actor: Passenger
Entry conditions: Passenger standing in front of ticket distributor.
Passenger has sufficient money to purchase ticket.
Event flow:
1. Passenger selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. Passenger inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

The <<extend>> Relationship
<<extend>> relationships represent exceptional or rarely invoked cases.
The exceptional event flows are factored out of the main event flow for clarity.
Use cases representing exceptional flows can extend more than one use case.
The direction of a <<extend>> relationship is towards the extended use case.

The <<include>> Relationship
An <<include>> relationship represents behavior that is factored out of the use case.
An <<include>> represents behavior that is factored out for reuse, not because it is an exception.
An <<include>> relationship arrow starts from the including use case (unlike <<extend>> relationships).
**Data Types and Instances**

In programming languages, **data types** are used as concepts:

- A data type has a **name** for identification,
- a data type has a **purpose** (structure and operations on its members),
- a data type has **members**.

An abstract data type is a data type whose structure and implementation is hidden from the rest of the system.

An instance is any member of a specific data type.

---

**Classes**

In object-oriented programming languages, **classes** are used as concepts (abstractions).

Classes are like abstract data types in terms of **encapsulating** structure and behavior.

But unlike abstract data types, classes can be derived from other classes by using **generalization**.

Imagine that you developed a watch that can also be used to surf the web.

Then the class **WebWatch** would be a refinement of the class **Watch**.

---

**Classes**

This type of relationship between two classes is called **generalization**.

The base class (Watch) is called the **superclass**, and the refined class is called the **subclass** (WebWatch).

The subclass refines the superclass by adding attributes and operations (or **modifying** existing ones).

In **UML class diagrams**, classes and objects are shown as boxes with three compartments containing:

- the **name** of the class or object (object:Class),
- its **attributes**,
- its **operations**.

---

**Classes**

An object is an **instance** of a class; each object belongs to **exactly one** class.

In UML, names of instances are underlined.

---

**Class Diagrams**

Class diagrams represent the structure of the system.

Classes specify the common structure and behavior of a set of objects.

**Objects**

- are instances of classes,
- are created, modified, and destroyed during the execution of the system,
- have a state.

Class diagrams describe the system in terms of objects, classes, attributes, operations, and their associations.
Classes
A class represents a concept.
A class encapsulates states (attributes) and behavior (operations).
Each attribute has a type.
Each operation has a signature (the tuple made out of the types of its parameters and the type of the return value)
The class name is the only mandatory information.

Instances
An instance represents a phenomenon.
The name of an instance is underlined and can contain the class of the instance.
The attributes are represented with their values.

Associations
Associations denote relationships between classes.
The multiplicity of an association end denotes how many objects the source object can legitimately reference.

Class Diagrams
Example: Class diagram for a simple watch.

Aggregation
An aggregation is a special case of association denoting a "consists of" hierarchy.
The aggregate is the parent class, the components are the children class.

Generalization
Generalization relationships denote inheritance between classes.
The child classes inherit the attributes and operations of the parent class.
Generalization simplifies the model by eliminating redundancy.
Sequence Diagrams

Sequence diagrams show the communication among objects. The objects involved in a use case are called participating objects.

A sequence diagram represents the interactions that take place among these objects. In a sequence diagram,

- columns represent timelines of actors and objects,
- labeled arrows represent stimuli (messages) that are sent from an actor to an object or between objects.

Sequence Diagram Observations

- UML sequence diagrams represent behavior in terms of interactions.
- Complement the class diagrams which represent structure.
- Useful to find participating objects.
- Time consuming to build but worth the investment.

Statechart Diagrams

Statechart diagrams describe the behavior of an individual object like a finite state automaton, that is,

- a finite number of states that the object can assume,
- transitions between these states.

Here, a state is a particular set of values for an object. States are represented by ovals.

A transition represents a state change that occurs under a particular condition.

Transitions are represented by arrows that are labeled with the corresponding condition.
**Statechart Diagrams**

**Example:** Statechart diagram for SimpleWatch.

```
State: BlinkHours
  Transition: button1&2Pressed
  Action: IncrementHours

State: BlinkMinute
  Transition: button1&2Pressed
  Action: IncrementMinute

State: BlinkSecond
  Transition: button1&2Pressed
  Action: IncrementSecond

State: StopBlinking
```

**Activity Diagrams**

An activity diagram describes a system in terms of activities. Activities are shown as states that represent the execution of a set of operations. After the completion of one or more activities, other activities are started. Activity diagrams are similar to flowchart diagrams. They represent activities as ovals and transitions between activities as arrows.

**Example:** Activity diagram for incident management

```
Open Incident
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate Resources</td>
</tr>
<tr>
<td>Coordinate Resources</td>
</tr>
<tr>
<td>Archive Incident</td>
</tr>
<tr>
<td>Document Incident</td>
</tr>
</tbody>
</table>
```

Thick bars represent the synchronization of the control flow.