Introduction to Object-Oriented Programming

- Objects and classes
- Abstract Data Types (ADT)
- Encapsulation
Pure Object-Oriented Languages

Five rules [Source: Alan Kay]:

- Everything in an object.
- A program is a set of objects telling each other what to do by sending messages.
- Each object has its own memory (made up by other objects).
- Every object has a type.
- All objects of a specific type can receive the same messages.

Java breaks some of these rules in the name of efficiency.
The Object Concept

- An object is an *encapsulation* of data.

- An object has
  - identity (a unique reference),
  - state, also called characteristics
  - behavior

- An object is an instance of an *abstract data type*.
- An abstract data type is implemented via a *class*. 
Abstract Data Type (ADT)

- An ADT is a collection of objects (or values) and a corresponding set of methods.
- An ADT encapsulates the data representation and makes data access possible at a higher level of abstraction.

- Example 1: A set of vehicles with operations for starting, stopping, driving, get km/liter, etc..
- Example 2: A time interval, start time, end time, duration, overlapping intervals, etc.
Encapsulation and Information Hiding

- Data can be encapsulated such that it is invisible to the "outside world".
- Data can only be accessed via methods.
Encapsulation and Information Hiding, cont.

- What the "outside world" cannot see it cannot depend on!
- The object is a "fire-wall" between the object and the "outside world".
- The hidden data and methods can be changed without affecting the "outside world".

![Diagram of encapsulation]

**An object**

- Hidden data and methods
- Visible data and methods

**Client interface**
## Class vs. Object

<table>
<thead>
<tr>
<th>Class</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>• A description of the <em>common properties</em> of a set of objects.</td>
<td>• A representation of the <em>properties</em> of a single instance.</td>
</tr>
<tr>
<td>• A concept.</td>
<td>• A phenomenon.</td>
</tr>
<tr>
<td>• A class is a part of a program.</td>
<td>• An object is part of data and a program execution.</td>
</tr>
<tr>
<td>• Example 1: Person</td>
<td>• Example 1: Bill Clinton, Bono, Viggo Jensen.</td>
</tr>
<tr>
<td>• Example 2: Album</td>
<td>• Example 2: A Hard Day's Night, Joshua Tree, Rickie Lee Jones.</td>
</tr>
</tbody>
</table>
Type and Interface

• An object has type and an interface.

<table>
<thead>
<tr>
<th>Account</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance()</td>
<td></td>
</tr>
<tr>
<td>withdraw()</td>
<td></td>
</tr>
<tr>
<td>deposit()</td>
<td>Interface</td>
</tr>
</tbody>
</table>

• To get an object  
  `Account a = new Account();`

• To send a message  
  `a.withdraw();`
Instantiating Classes

- An instantiation is a mechanism where objects are created from a class.
- Always involves storage allocation for the object.
- A mechanism where objects are given an initial state.

**Static Instantiating**
- In the declaration part of a program.
- A static instance is implicitly created

**Dynamic Instantiating**
- In the method part of a program.
- A dynamic instance is created explicitly with a special command.
Interaction between Objects

- Interaction between objects happens by *messages* being send.
- A message activates a method on the calling object.

- An object O1 interacts with another object O2 by calling a method on O2 (must be part of the client interface).
  - "O1 sends O2 a message"
- O1 and O2 must be *related* to communicate.
- The call of a method corresponds to a procedure call in a non-object-oriented language such as C or Pascal.
Phenomenon and Concept

- A *phenomenon* is a thing in the "real" world that has individual existence.
- A *concept* is a generalization, derived from a set of phenomena and based on the common properties of these phenomena.

- Characteristics of a concept
  - A name
  - *Intension*, the set of properties of the phenomenon
  - *Extension*, the set of phenomena covered by the concept.
Classification and Exemplification

- A *classification* is a description of which phenomena that belongs to a concept.
- An *exemplification* is a phenomenon that covers the concept.
Aggregation and Decomposition

- An *aggregation* consists of a number of (sub-)concepts which collectively is considered a new concept.
- A *decomposition* splits a single concept into a number of (sub-)concepts.
Aggregation and Decomposition, Example

- Idea: make new objects by combining existing objects.
- *Reusing the implementation!*

<table>
<thead>
<tr>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>start()</td>
</tr>
<tr>
<td>drive()</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Gearbox</td>
</tr>
<tr>
<td>Doors[4]</td>
</tr>
</tbody>
</table>
Generalization and Specialization

- **Generalization** creates a concept with a broader scope.
- **Specialization** creates a concept with a narrower scope.
- **Reusing the interface!**

![Diagram showing generalization and specialization concepts]

- **Concept A** specializes into **Concept B**.
- **Concept C** generalizes into **Concept D**.

**Vehicle**

- **Car**: Hatchback, Station car, Sedan
- **Truck**: Pickup
Generalization and Specialization, Example

- *Inheritance*: get the interface from the general class.
- Objects related by inheritance are all of the same type.

```
<table>
<thead>
<tr>
<th>Shape</th>
<th>draw()</th>
<th>resize()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circle</td>
<td>draw()</td>
<td>resize()</td>
</tr>
<tr>
<td>Line</td>
<td>draw()</td>
<td>resize()</td>
</tr>
<tr>
<td>Rectangle</td>
<td>draw()</td>
<td>resize()</td>
</tr>
<tr>
<td>Square</td>
<td>draw()</td>
<td>resize()</td>
</tr>
</tbody>
</table>
```
Code Example

```java
void doSomething (Shape s){
    s.draw;
    s.resize;
}
Circle c = new Circle();
Line l = new Line();
Rectangle r = new Rectangle ();
doSOMething (c);                 // dynamic binding
doSOMething (l);
doSOMething (r);
```

- **Polymorphism**: One piece of code works with many all shape object.
- **Dynamic binding**: How polymorphism is implemented.
Structuring by Program or Data?

- What are the actions of the program vs. which data does the program act on.
- *Top-down*: stepwise program refinement
- *Bottom-up*: Focus on the stable data parts then add methods,

Object-oriented programming is bottom-up. Programs are structure with outset in the data.
Java Program Structure

// comment on the class
public class MyProg {
    String s = "Viggo";

    /**
     * The main method
     */
    public static void main (String[] args){
        // just write some stuff
        System.out.println ("Hello World");
    }
}

variable

method header

method body
Byte Code vs. Executable

MyProg.java

```
javac MyProg.java
```

Java Class File
MyProg.class
Portable Byte Code
Java Virtual Machine
Operating System

MyProg.cpp

```
gcc MyProg.cpp -o myprog.exe
```

Executable myprog.exe
Operating System
History of Java

- 1990 Oak (interactive television, big failure)
- 1994 Java (for the Internet)
  - Main feature: "Write Once, Run Any Where"
    => wrap the operating system so they all look the same.

Designed for

- A fresh start (no backward compatibility)
- "Pure" OOP: C++ Syntax, Smalltalk style
- Improvements over C++ much harder to write a bad program
- Internet programming
  - Very hard to create a virus
  - Run in a web browser (and at the server)
- There is a speed issue (Java 1.3 and up much better)
Difference from C/C++

- Everything resides in a class
  - variables and methods
- No global variables or methods
- No local static variables
- No separation of declaration and implementation (no header files).
- No explicit pointer operations (uses references)
- No preprocessor (but something similar)
- Has fewer "dark corners"
- Has a much larger standard library
Summary

- Classes are "templates".
- All objects are instances of classes.
- An ADT is implemented in a class

- Encapsulation
  - Key feature of object-oriented programming
  - Separation of interface from implementation
  - It is not possible to access the private parts of an object