

CSC 142

Java objects: a first view [Reading: chapter 1]

CSC 142 B 1

What is an object? (1)

- An example: a vending machine
 - It has things: candy, cans, coffee, money, ...
 - Some information is public: number of candy bars...
 - Some information is private: the amount of money...
- The vending machine can perform actions:
 - accept money, give change, give food, refrigerate...
 - Some actions are available to anyone, others require special access (repair person)

CSC 142 B 2

What is an object? (2)

- The machine provides an interface to its behavior (button panel). The customer doesn't need to know the internal workings of the machine.
- There can be many identical machines all based on the same design. However, each machine has its own identity (some are out of order, some have more candy, etc...).
- Java allows us to reproduce this view on the computer.

CSC 142 B 3

An object in Java

- An object is an instance of a **class**.
- The **class** is the blueprint. It describes
 - The data contained in the object. Some are **private**, some are **public**.
 - The actions that the object can perform. Some actions are available to anyone (**public** methods). Others require special access (**private** methods).
- The interface is made of the **public** data and methods. It describes what the object can do for us. We don't need to know how the object does it (the details are hidden = **private**).

CSC 142 B 4

Why using objects?

- It corresponds to the way we view the world.
 - A plane has reactors, two wings... It can fly, take off, land, carry passengers...
 - We can use the same framework to solve problems on the computer.
- Objects enhance software reusability.
 - Once a class is defined, we can use it over and over. We will do so with many classes of the Java API.
 - As long as the interface is unchanged, the inner workings (=implementation) of the class can be modified without requiring any changes on the part of the users of the class.

CSC 142 B 5

Using objects in Java

- An example: Display a circle within a graphics window.
- Where are the objects?
 - We want an object that has two objects, a circle and a graphics window. The object should put the circle in the graphics window.
 - In Java, do so by writing the blueprint of the object (=class). Then, to get the object, instantiate the class.
 - What about creating a circle and a graphics window? Difficult from scratch, easy if we use code already written in libraries.

CSC 142 B 6

Interlude: Java libraries

- Library
 - A set of classes already written ready to use.
 - In our example, we want a library that has classes (=blueprints) for a graphics window and a circle. Use the UW library.
 - Java has an enormous amount of libraries.
 - Programmers can reuse code already written to write their programs (fast, easy and less likely to have bugs).
 - Important to know what is available
 - A library is often called an Application Programmer Interface (=API).

CSC 142 B 7

WindowWithCircle class

- Name the class
 - Use a **meaningful** name, e.g., WindowWithCircle
 - The style in Java is to capitalize each letter of each word of the class name (do so as well).
 - A name can contain letters, digits (e.g. CarModel12), the underscore (_), or currency symbols (\$, £, ¥, ...).
 - A name can't start with a digit (e.g. 1NoGood is not a valid class name).
 - A name can't be a reserved java keyword (e.g. class).
 - In practice, a name can be as long as you want.
- case sensitive (MyClass ≠ Myclass).

CSC 142 B 8

Instance fields

- Name the objects needed to build a WindowWithCircle object = **instance fields**
 - A GWindow object, e.g., **window**
 - A Oval object, e.g., **circle**
 - (GWindow and Oval are class names from the CSE142 UW library)
- Naming instance fields
 - same rules as for class names, except that the first letter is lowercase
 - e.g., aBlueCircle, aDialogBox

CSC 142 B 9

Instance methods

- Name the actions that a WindowWithCircle object can perform = **instance methods**
 - Namely, create a graphics window and a circle. Put the circle in the graphics window.
 - Do it when building a WindowWithCircle object.
 - Done in a special method, called the **constructor**. The constructor of a class has the same name as the class name, WindowWithCircle (no other choice).
- No other methods needed here.

CSC 142 B 10

UML representation

- UML: Unified Modeling Language
 - a set of graphical rules to display a design when programming in an OOP context.
- WindowWithCircle class diagram

```
classDiagram
    class WindowWithCircle {
        GWindow window
        Oval circle
        WindowWithCircle()
    }
```

 - More rules in UML (see later).

CSC 142 B 11

Code for WindowWithCircle

- In a file that **must** be named WindowWithCircle.java, write

```
import uwse.graphics.*; //uw graphics library
public class WindowWithCircle{
    // instance fields
    private GWindow window;
    private Oval circle;

    // Only one instance method: the constructor
    public WindowWithCircle(){
        /* Create the window and the circle
        Put the circle in the window */
        window = new GWindow();
        circle = new Oval();
        window.add(circle);
    }
}
```

 - *one line comment* (points to `//uw graphics library`)
 - *every java statement ends with a semi colon* (points to `private GWindow window;`)
 - *multiline comment* (points to `/* Create the window and the circle Put the circle in the window */`)

CSC 142 B 12

Code organization

- Code is written inside of blocks {} that are class definitions
 - `public class` WindowWithCircle {
 / my code is here */*}
 - `public` means that the class is available to everyone (No privileged access is required. More on this later).
- The file that contains the definition of the `public class` WindowWithCircle **must be** named: WindowWithCircle .java (case sensitive!)
- Only one public class** per java file

CSC 142 B 13

The `import` statement

- To access the content of libraries
- `import uwcse.graphics.*` means:
 - we can use any class listed in the folder graphics which is in the folder uwcse.
 - uwcse.graphics is called a **package**.
- To limit the access to GWindow and Oval **only**
`import uwcse.graphics.GWindow;`
`import uwcse.graphics.Oval;`
 - More specific for the reader. But, requires several import statements.
- Using `import` is optional. If omitted, need to write the full name in the code, e.g. uwcse.graphics.GWindow instead of just GWindow.

CSC 142 B 14

Comments

- Ignored by the computer. Essential for the reader.
- 2 types of comments: // and */* */*
 - Everything between // and the end of the line is a comment
 - Everything between */** and **/* is a comment. It can be used on several lines. You can't nest these comments (*/* blabla /* blabla */ blabla */* gives an error)
- Examples
 - // this is a comment*
 - /* this is a comment that can be written on several lines */*
- Also javadoc comments */*** and **/*

CSC 142 B 15

Instance fields

- Declaration
 - e.g., `private GWindow window ;`

access modifier type identifier
 - identifier: the name of the instance field
 - access modifier: `private` (the identifier can only be used in the class, i.e., within the block {} of the class). `public` (the identifier can be used outside the class). More on this later.
 - type: the class name of the identifier.
- A declaration **doesn't** create an object. It just creates a name for an object.

CSC 142 B 16

Instantiating a class

- To create an object, e.g. a GWindow, write
`window = new GWindow();`
- An object of type GWindow is created by executing the statements listed in the constructor of the GWindow class.
- Now, window refers to an actual object (window refers to a chunk of memory that contains information about a GWindow).
- In UML (for object diagrams)

`window :GWindow`

CSC 142 B 17

Constructor: a first view

- A special instance method of a class executed when (and **only** when) an instance of the class is created (using `new`), as in
`circle = new Oval();`
// execute constructor of Oval class
- It must have the same name as the class
- Syntax (e.g. for WindowWithCircle)
`public windowWithCircle(){/*write code here*/}`
- `public` means that anyone can instantiate the class. What if we used `private`?
- Can have multiple constructors in the same class (see later).

CSC 142 B 18

WindowWithCircle constructor(1)

- Create a GWindow and an Oval: OK, use `new`

```
window = new GWindow();  
circle = new Oval();
```
- Put the circle in the window: need to know what a GWindow and an Oval can do.
- Whenever using a library, read the documentation. It describes every public member of the class (=interface for the user of the class).
- Available directly inside the java IDE or online.

CSC 142 B 19

WindowWithCircle constructor(2)

- GWindow lists the instance method `add`, to add an item to a GWindow object.
- Use the dot notation to access the instance method via the reference to the object, i.e.

```
window.add(circle);
```
- Note that `add` takes an input parameter (namely `circle`). Our next chapter will describe such methods.

CSC 142 B 20

Running the code

- In BlueJ, instantiate the class by right clicking on the class icon and select `new WindowWithCircle()`.
 - The object appears on the object bench.
 - By right clicking on the object icon, you can inspect the object (list its public and private instance fields) or list the public methods of the object. (see lab).

CSC 142 B 21

A few questions

- Change the color of the circle?
- Displaying the circle at another location within the graphics window?
- Displaying the window at another location on the computer screen?
- All of the above can be done. It requires using the right methods from the GWindow or Oval classes (try it!).

CSC 142 B 22