## CSC 142

Java objects: a first view

[Reading: chapter 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)

## 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.

## 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).

## Why using objects?

- It corresponds to the way we view the world.
  - A plane has engines, 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.

## Using objects in Java

- An example: a person defined by a name and an age.
- Where are the objects?
  - The person is the object. It is defined in terms of other objects = the name and age.
  - In Java, do so by writing the blueprint of the object (=class). Then, to get the object, instantiate the class.
  - What about defining a name and an age? Age not too difficult just a number. Name more difficult from scratch, easier if we use code already written in libraries.

## Interlude: Java libraries

- Library
  - A set of classes already written ready to use.
  - To represent a name and an age. Use the String class for the name and possibly the Integer class for the age (though see later)
  - 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).

## Person class

- Name the class
  - Use a meaningful name,e.g., Person
  - The style in Java is to capitalize each letter of each word of the class name (do so as well), e.g. NeutronStar → Pascal case or upper camel case.
  - 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).
  - A name can be as long as you want.
  - case sensitive (MyClass ≠ Myclass).

## Instance fields

- Name the objects needed to build a Person object = instance fields
  - A String object, e.g., **`name`**
  - An Integer object, e.g., **`age`**
  - In practice, use an `int` for **`age`**. See later.
  - String and Integer are class names from the Java library.
- Naming instance fields (lower camel case)
  - same rules as for class names, except that the first letter is lowercase
  - e.g., aBlueCircle, aDialogBox

## Constructor(s)

- The constructor is a special function called to create an object as described by the class.
  - The constructor of a class has the same name as the class, i.e. Person (NO other choice).
  - You may define as many constructors as you want.
  - Different constructors take different parameters.
  - For instance we may define the Person constructor as taking a String and an int.

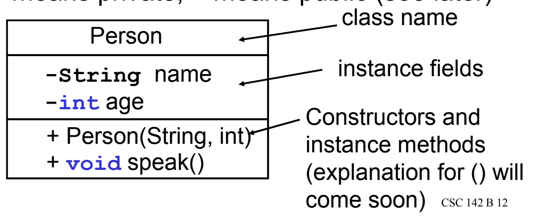
## Instance methods

- Name the actions that a Person object can perform = instance methods
  - Namely, a Person object can speak.
  - Can have as many methods as needed.
  - Method names follow the same rules as variable names, i.e. lower camel case.

## Class diagram: UML representation

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

  - means private, + means public (see later)



## Code for Person

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

```
public class Person{                                  one line comment
  // a person is defined by a name and age
  private String name;   every java statement
  private int age;        ends with a semi colon

  /* Creates a person with the given age and
  name */
  public Person(String n, int a){
   name = n;                                 multiline comment
   age = a;
  }
   public void speak() {
     System.out.println("Hello, I am " + name +
     ". My age is " + age + ".");
}
```

## Code organization

- Code is written inside of blocks {} that are class definitions
  - public class Person {
    /* 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 Person must be named: Person .java (case sensitive!)
- Only one public class per java file

## 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 */

## Instance fields

- Declaration
  - e.g., `private` `String` `name ;`

    *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 bloc {} 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.

## Running the code: client code

- Write code that creates a person and makes it speak
  - Write a client class with a main method

```
public class PersonUser {
 public static void main(String[] args) {
   Person p = new Person("Sara", 25);
   p.speak();
 }
}
```

## Instantiating a class

- To create an object, e.g. a Person, write

```
Person p = new GWindow("Sara", 25);
```

- An object of type Person is created by executing the statements listed in the constructor of the Person class.
- Now, p refers to an actual object (p refers to a chunk of memory that contains information about a Person).
- In UML (for object diagrams)

## Calling a method

- To make the person speak, write

```
p.speak();
```

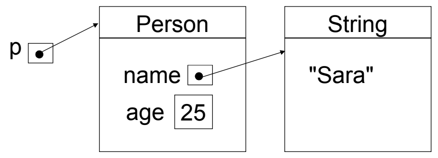
- The speak method of the Person class is called with the object referred to by p.

## Object diagram

- An object diagram represents the objects currently in memory at some point of the execution of a program.
- e.g. just after the line

```
Person p = new Person("Sara", 25);
```

p points to an object of type Person.



## "Our" object diagram conventions

- A dot ● means an address
- An object in memory is represented as a table with two rows:
  1st row: object type
  2nd row: list of the instance fields (some of which may point to more objects)
- To simplify, display some objects with just their contents (e.g. "Sara" instead of listing the fields of the String class)
- Sometimes write the first row as Person: p instead of just Person

## Another example with graphics

- 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.

## 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).

## 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).

## Instance fields

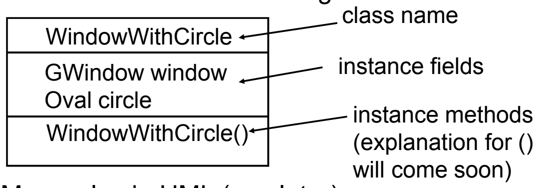
- Name the objects needed to build a WindowWithCircle object = instance fields
  - A GWindow object, e.g., **`window`**
  - A Oval object, e.g., **`circle`**
  - (GWindow and Oval are class names from the CSE142 UW library)
- Naming instance fields
  - same rules as for class names, except that the first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

## 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.

## UML representation

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



- More rules in UML (see later).

## Code for WindowWithCircle

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

```
import uwcse.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*

*every java statement ends with a semi colon*

*mutiline comment*

## 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

## 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;
```

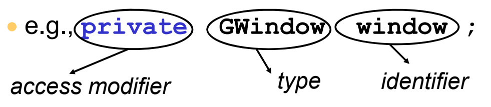
  - More specific for the reader. But, requires several import statements.
- Using import is optional. If omitted, need to write the full name in the code, e.g. uwcse.graphics.GWindow instead of just GWindow.

## 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 */

## Instance fields

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



  - identifier: the name of the instance field
  - access modifier: private (the identifier can only be used in the class, i.e., within the bloc {} 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.

## 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

## 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).

## 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.

## 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.

## A few questions

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