CSEN 202 – Introduction to Computer Programming

Lecture 7: Classes and objects II

Prof. Dr. Slim Abdennadher,
Dr. Wael Abouelsaadat and
Dr Mohammed Abdel Megeed Salem
slim.abdennadher@guc.edu.eg

German University Cairo, Faculty of Media Engineering and Technology

April 15 - April 20, 2017

Classes and objects I

- An object is a record of various data fields that carries its own functionality in terms of various methods.
- A class is a factory for objects.
- Any static field or method (defined with the keyword `static`) belongs to the class.
- Any dynamic (*i. e.*, non-static) field or method is replicated on each individual object.

Person

- All persons are described by a common set of properties or fields (instance variables)
  - Name
  - Year of birth
- The object type is based on the names and types of its fields.
- The main role of classes is to define types of objects.

Example

```
public class Person {
   String name;
   int yearOfBirth;
}
```

The constructor

- Each instance of this class (object of this type) will have its own copies of the instance variables (field values)
- Create objects of a given class with appropriate field values

```
public class Person {
  String name;
  int yearOfBirth;
  public Person (String n, int y) {
    name = n;
    yearOfBirth = y;
  }
}
```

This

- To distinguish between arguments and fields, the keyword **this** can be used:

```
public Person (String n, int y) {
  name = n;
  yearOfBirth = y;
}

public Person (String name, int yearOfBirth) {
  this.name = name;
  this.yearOfBirth = yearOfBirth;
}
```

- Note: "name" (argument) *vs.* "**this**.name" (field) and "yearOfBirth" (argument) *vs.* "**this**.yearOfBirth" (field)

Making a (virtual) person

- Declare a variable of appropriate type to hold the `Person` object.
- Call the constructor for `Person` with appropriate arguments.

```
Person ph = new Person("Haythem", 1970);
```

Reading and object's data

```
Person ps = new Person("Slim", 1967);
```

- `ps.name` ⇒ `"Slim"`
- `ps.yearOfBirth` ⇒ `"1967"`

```
Person pg = new Person("Georg", 1973);
```

- `pg.name` ⇒ `"Georg"`
- `pg.yearOfBirth` ⇒ `"1973"`

Instance methods

An instance method is a subroutine or function designed to work on the current object.

- A method to change the person's name:

```
public void setName(String name) {
  this.name = name;
}
```

- A method to get the person's name:

```
public String getName() {
  return name;
}
```

Example—Instance methods

An instance method is a subroutine or function designed to work on the current object.

- A method to display the name and the year of Birth of a person:

```
public void display() {
  System.out.println("Name:\t\t " + name);
  System.out.println("Year of birth:\t " + yearOfBirth);
}
```

Example—Invoking an instance method

Instance methods apply to objects of the class containing the methods

```
public static void main(String[] args) {
  Person ps = new Person("Slim", 1967);
  Person pg = new Person("Georg", 1973);
  Person pc = new Person("Christian", 1975);
  Person ph = new Person("Haythem", 1970);

  /* Testing display() */
  System.out.println("Display pg and ps:");
  pg.display();
  ps.display();
  System.out.println();
}
```

Class variables

- We want to keep a track of every instance of a Person class.
- If we could have a variable that was visible to every instance, we could increment it every time.
- If we declare an instance variable as `static`, it becomes a class variable, and can be seen and modified by all instances.

Class variables

Example:

```
public class PersonID {
  private static int nextID = 1;

  private String name;
  private int yearOfBirth;
  private int iD;

  public PersonID (String name, int yearOfBirth) {
    this.name = name;
    this.yearOfBirth = yearOfBirth;
    this.iD = nextID++;
  }
}
```

Class methods

- An instance method is a method that is invoked from a specific instance of a class that performs some action related to that instance.
- A class method is not necessarily associated with a particular object and need not be invoked from an open object.
  Class methods are declared with the `static` keyword.

Class methods

Example:

```
public static int getNextID() {
  return nextID;
}
```

Access and invocation

```
public class PersonIDTester {

  public static void main(String[] args) {
    PersonID ps = new PersonID("Slim", 1967);
    ps.display();
    System.out.println("The next ID is " +  PersonID.getNextID());
    PersonID pg = new PersonID("Georg", 1973);
    pg.display();
    System.out.println("The next ID is " + PersonID.getNextID());
  }
}
```

- Instance methods are invoked through an object (*e. g.*, `ps.display()`)
- Class methods are invoked throuhg the class name (*e. g.*, `PersonID.getNextID()`)

Value *vs.* reference

Note: Variables for object-types are references only!

- A variable of primitive type marks a memory cell that contains a value. For example:

```
int i = 5; // create an int (primitive type)
```

Then `i` is the name of a memory-cell containing 5:

Value *vs.* reference

Note: Variables for object-types are references only!

- However a variable of an object type marks a memory cell that contais a reference (address) to the actual object. For example:

```
Person pg = new Person("Georg", 1973);
```

Then `pg` is the name of a memory-cell containing the address of the object:

`pg` $\rightsquigarrow$ `address` $\rightarrow$ `object`

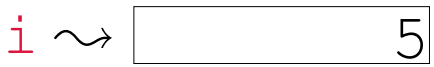
Value *vs.* reference

Note: Variables for object-types are references only!

- Therefore, primitive values can be copied (cloned) easily:

```
int j = i; // create a copy of i (primitive type)
```

Then:



and

Value *vs.* reference

Note: Variables for object-types are references only!

- The value of `j` can naturally be changed without influencing `i`:

```
j = 6; // Change the value of j
```

Then:

`i` ⇝ 5

and

`j` ⇝ 6

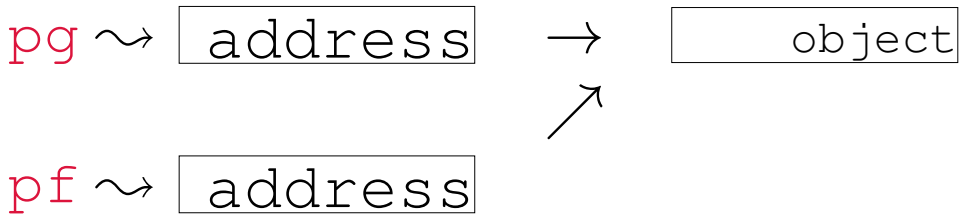
Value *vs.* reference

Note: Variables for object-types are references only!

- However, if an object type is copied in that same way, the address only is cloned:

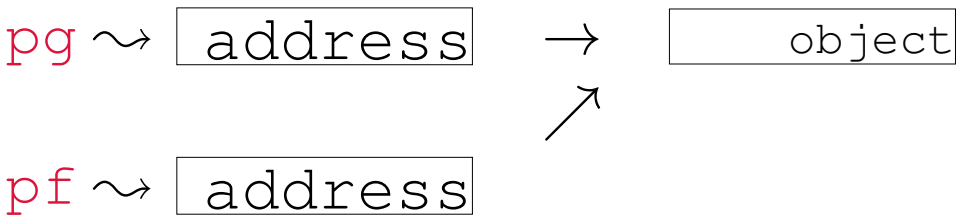
```
Person pf = pg; // create a copy of pg (object)
```

Then:



Value *vs.* reference

Note: Variables for object-types are references only!



- `pf.setName("Faruk"); // Change the value of pf (!)`

Will change the field `name` on both variables!!

Value *vs.* reference

```
public static void main(String[] args) {
  Person ps = new Person("Slim", 1967);
  Person pg = new Person("Georg", 1973);
  Person pf = pg; // create a copy of pg (object)
  pf.setName("Faruk"); // Change the value of pf (!)
}
```

What are the actual values of `pg.name` and `pf.name` now?

The null-reference

`null`

The line

```
Person pa;
```

is equivalent to

```
Person pa = null;
```

- An object variable that is not initialized will be set to a default (non-existing) address called **`null`**.
- Accessing a field or calling a method on a **`null`**-reference leads to a run-time error!

Equality and identity

- Create two objects with the same data:

```
Person ps = new Person("Slim", 1967);
Person pt = new Person("Slim", 1967);
```

- What is the output of

```
if (ps == pt)
  System.out.println("ps and pt refer to the same object.");
else
  System.out.println("ps and pt refer to different objects.");
```

Equality and identity

- Create an object and a copy:

```
Person pg = new Person("Georg", 1973);
Person pf = pg;
```

- Change a field on one copy (!!!)

```
pf.setName("Faruk");
```

- What is the output of

```
if (pg == pf)
  System.out.println("pg and pf refer to the same object.");
else
  System.out.println("pg and pf refer to different objects.");
```

To check equivalence of objects you have to implement an `equals()` method!

Hints on designing a class

- Step 1: Find out what you are asked to do with an object of the class.

Example (bank account)

Suppose you are asked to implement a class `BankAccount`.
Operations:

- deposit money
- withdraw money
- get balance

Hints on designing a class

- Step 2: Find names for the methods.

Example (bank account)

```
BankAccount harrysChecking = new BankAccount();
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
harrysChecking.getBalance();
```

Hints on designing a class

- Step 3: Determine instance variables

Example (bank account)

```
private double balance;
```

Hints on designing a class

- Step 4: Determine constructors

Example (bank account)

- Open a back account

```
public BankAccount () {
  balance = 0;
}
```

- Open a back account with an initial balance

```
public BankAccount (double initialBalance) {
  balance = initialBalance;
}
```

Hints on designing a class

- Step 5: Implement the methods.
- Step 6: Test your class

```
public class BankAccountTester {

   public static void main(String[] args) {
      BankAccount harrysChecking = new BankAccount();
      harrysChecking.deposit(2000);
      harrysChecking.withdraw(500);
      System.out.println(harrysChecking.getBalance());
   }
}
```

Designing a simple class

A point on a plane is given by two coordinates x and y in a fixed frame of reference

```
public class Point {
  double x; // first coordinate
  double y; // second coordinate

  /** Create a point with the given coordinates */
  public Point (double x, double y) {
    this.x = x;
    this.y = y;
  }
}
```

Designing a simple class

An operation is to move the point:

```
void move (double dx, double dy) {
  this.x += dx;
  this.y += dy;
}
```

Building on

A circle is defined by its center (a point) and its radius.

```
public class Circle {
  Point center;
  double radius;

  /** Create a circle given center and radius */
  public Circle (Point center, double radius) {
    this.center = center;
    this.radius = radius;
  }
```

Building on

Testing the circle class:

```
public class CircleTester {

  public static void main(String[] args) {
    Point p = new Point (1, 2);
    Circle c = new Circle (p, 5);
    System.out.println (c.center.x);
  }
}
```

Multiple constructors

- It is often convenient to construct objects in a variety of ways
- A constructor can be overloaded (*i. e.*, multiple constructors can be distinguished by argument numbers and types)

Example (Constructor with a given point)

```
public Circle (Point center, double radius) {
  this.center = center;
  this.radius = radius;
}
```

Example (Constructor given center coordinates)

Multiple constructors

```
/** Create a circle given center and radius */
public Circle (Point center, double radius) {
  this.center = center;
  this.radius = radius;
}

/** Create a circle given center coordinates
    and radius */
public Circle (double x, double y, double radius) {
  this (new Point (x, y), radius);
}
```

Coming up

- Next topic: Arrays and array algorithms