CSEN202 – Introduction to Computer Programming

Topics:

Objects and Classes II

Prof. Dr. Slim Abdennadher

15.05.2008

Objects and Classes – Example

- 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

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

Constructing Objects – Example

- 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 yOfB) {
    name = n;
    yearOfBirth = yOfB;
  }
}
```

Making a (virtual) Person

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

```
Person pm = new Person("Tony", 1953);
```

Reading an object’s data

```
Person pm = new Person("Tony", 1953);
```

`pm.name` $\Rightarrow$ `"Tony"`

`pm.yearOfBirth` $\Rightarrow$ 1953

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

`slim.name` $\Rightarrow$ `"Slim"`

`slim.yearOfBirth` $\Rightarrow$ 1967

Instance Methods (I)

- 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 newName){
   name = newName; }
```

- A method to get the person's name:

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

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

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

Instance Methods (II)

- Instance Methods apply to objects of the class containing the methods

```
    public static void main(String[] args){
     Person pm = new Person("Tony", 1953);
     pm.display();
     pm.setName("Williams");
     pm.display();
}
```

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.

-
  ```
  public class Person {
    String name;
    int yearOfBirth;
    static int number;

    public Person(String n, int yOfB) {
      name = n;
      yearOfBirth = yOfB;
      number++;
    }
  }
  ```

Class Methods

- **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.

```
public  static int totalNumberofPersons() {
   return number;
}
```

Designing a simple class

- A point on the plane is given by its coordinates `x`, `y` in a fixed frame of reference

```
class Point {
  // First coordinate.
  double x;
  // Second coordinate.
  double y;
  // Create a new point
  Point(double anX, double aY) {
    x = anX;
    y = aY;
  } }
```

- **Method:** Move the point

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

Building on

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

```
class Circle {
  // The center of the circle
  Point center;
  // The radius of the circle
  double radius;

  // Create a  Circle instance
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  } }
```

- **Complex objects**:

```
Point  p = new Point(1,2);
Circle c =  new Circle(p,0.5);
System.out.println(c.center.x);    // 1.0
```

this in instance methods

- Within an instance method, `this` refers to the instance being operated on.

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

- Really means

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

Multiple Constructors

- It is often convenient to construct objects of a type in a variety of ways.
- **Constructor** selected by argument numbers and types

```
class Circle {
  Point center;
  double radius;
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  }

  Circle(double cx, double cy, double aRadius) {
    center = new Point(cx,cy);
    radius = aRadius;
  }
}
```

this in Constructors

- In a **constructor**, `this` can refer to another constructor for the same class

```
class Circle {
  Point center;
  double radius;
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  }
  Circle(double cx, double cy, double aRadius) {
    this(new Point(cx,cy),aRadius);
  }
}
```