THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

COMP 110
Introduction to Programming

Thursday October 30, 2014

Jay Aikat
Fall 2014
TR 9:30 – 10:45, GS-G100

Previous Class

- What did we discuss?

Announcements

- **Assignment4 – Due Thu, Oct 30 (still working on it?)**
- **Reading: 5.1, 5.2, 5.3**
- **Midterm – Q11 (15 or 16): +1 for all**

Example: Rectangle

```
public class Rectangle
{
    public int width;
    public int height;
    public int area;

    public void setDimensions(int newWidth,
        int newHeight){
        width = newWidth;
        height = newHeight;
        area = width * height;
    }

    public int getArea(){
        return area;
    }
}
```

```
Rectangle box = new Rectangle();
box.setDimensions(10, 5);
System.out.println(box.getArea());


// Output: 50


box.width = 6;
System.out.println("The rectangle
   with edges " + box.width + "
   and " + box.height + " has area
   size " + box.getArea());


// Output: The rectangle with
   edges 6 and 5 has area size 50

// Wrong answer!
```

Accessors and Mutators

- How do you access private instance variables?
- Accessor methods (a.k.a. get methods, **getters**)
  - Allow you to look at data in private instance variables
- Mutator methods (a.k.a. set methods, **setters**)
  - Allow you to change data in private instance variables

Example: Student

```
public class Student
{
    private String name;
    private int age;

    public void setName(String studentName) {
        name = studentName;
    }
    public void setAge(int studentAge) {
        age = studentAge;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
```

Mutators

Accessors

Private Methods

- **Why make methods private?**
- Helper methods that will only be used from inside a class should be private
  - External users have no need to call these methods

Constructors

- Create and initialize new objects
- Special methods that are called when (and **only** when) creating a new object

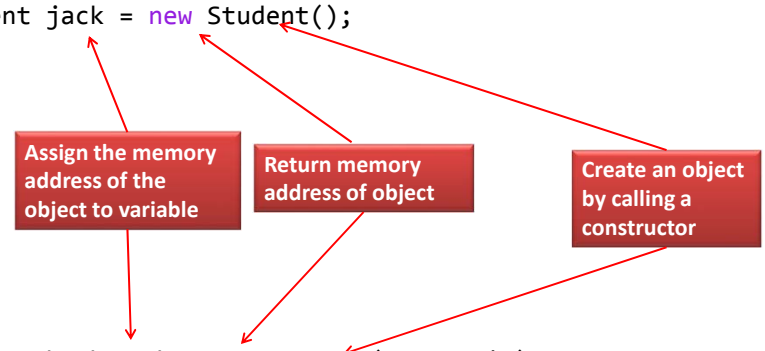
```
Student jack = new Student();
```

Calling a constructor

Creating an Object

Create an object *jack* of class *Student*

```
Student jack = new Student();
```




```
Scanner keyboard = new Scanner(System.in);
```

Create an object *keyboard* of class *Scanner*

Constructors

- Can perform any action you write into a constructor's definition
  - There are no specific rules about what's in a constructor
- Meant to perform initializing actions
  - Usually, initializing values of instance variables by the creator of the object

Similar to Setter Methods

- However, constructors *create* an object in addition to setting the values of instance variables
- Like methods, constructors can have parameters

Example: Pet Class

```
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet()
    {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public static void main(String[] args)
    {
                Pet p = new Pet();
        }
}
```

Default constructor

Call constructor

The Same as Initialization

```
public class Pet
{
    private String name = "No name yet.";
    private int age = 0;
    private double weight = 0;


    public static void main(String[] args)
    {
                Pet p = new Pet();
        }
}
```

Default constructor not declared – but still exists

Call default constructor (so an object is created)

Default Constructor

- Constructor that takes no parameters

```
public Pet()
{
    name = "No name yet.";
    age = 0;
    weight = 0;
}
```

- Java automatically defines a default constructor if you do not define any constructors
  - You've never written a constructor but you can still create objects

Constructor with Parameters

```
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet(String initName, int initAge, double initWeight)
    {
        name = initName;
        age = initAge;
        weight = initWeight;
    }

    public void setPet(String newName, int newAge, double newWeight)
    {
        name = newName;
        age = newAge;
        weight = newWeight;
    }
}
```

Another version of constructor that has parameters

A Closer Look

Same name as class name

```
public Pet(String initName, int initAge, double initWeight)
{
    name = initName;
    age = initAge;
    weight = initWeight;
}
```

Body

Parameters

No return type

Constructor with Parameters

- If you define at least one constructor, a default constructor will **not** be created for you
- Now you **must** create a Pet object like this:
  - `Pet odie = new Pet("Odie", 3, 8.5);`
  - `Pet odie = new Pet(); // WRONG! No default constructors!`

```
public class Pet {
    private String name;
    private int age;
    private double weight;
    public Pet(String initName, int initAge, double initWeight)
    {
        name = initName; age = initAge;   weight = initWeight;
    }
}
```


Multiple Constructors

- You can have several constructors per class
  - They all have the same name, just different parameters
    - Remember that the name is **the same as the class name**
  - The methods (with the same name) will be called according to its parameters

Multiple Constructors

```
public class Pet {
    private String name;
    private int age;
    private double weight;

    public Pet() {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public Pet(String initName, int initAge, double initWeight) {
        name = initName;
        age = initAge;
        weight = initWeight;
    }

         public static void main(String[] args)       {
                 Pet p = new Pet();
                 Pet q = new Pet("Garfield", 3, 10);
         }
}
```


Multiple Constructors

```
public class Pet {
    private String name = "No name yet.";
    private int age = 0;
    private double weight = 1; // The instance variables are initialized

    public Pet() {
        name = "No name yet.";
        age = 0;
        weight = 0;
    }

    public Pet(String initName, int initAge, double initWeight) {
        name = initName;
        age = initAge;
        weight = initWeight;
    }

    public Pet(String initName) {
        name = initName;
    }

    public static void main(String[] args) {
          Pet p = new Pet(); // p.weight is 0 – it is overwritten by constructor

          Pet q = new Pet("Garfield", 3, 10);

          Pet w = new Pet("Odie"); // w.weight is 1, as only one constructor
          //can be called. Variables will get initial value if not set in
          constructor.
    }
}
```

Calling a Constructor

- A constructor can be only called once when the object is created
  - `Pet odie = new Pet("Odie", 3, 8.5);`
- You can not invoke a constructor from an object
  - `odie.Pet("Odie", 3, 8.5);`
    `// Wrong! A constructor can not be invoked this way`
  - `odie.setPet("Odie", 3, 8.5);`
    `// Yes. You can use a setter instead`

Calling a Setter from the Constructor

```
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet(String initName, int initAge, double initWeight)
    {
        setPet(initName, initAge, initWeight);
    }

    public void setPet(String newName, int newAge, double newWeight)
    {
        name = newName;
        age = newAge;
        weight = newWeight;
    }
}
```

Initialization and Setting Instance Variables

- Initialization values give values to instance variables that are the same (or commonly the same) for all objects
- Constructors give values to instance variables that should be decided for each object
- Setters give values to instance variables that can be changed during time
  - If a value is never going to be changed, no setter is needed

Example: Initialize, Construct and Set

```
public class Pet {
    private String name;
    private int age = 0;
    // Age is always 0 (assuming newly-born pets are registered immediately)
    private double weight;

    public Pet(String initName, double initWeight){
        name = initName;
        weight = initWeight;
                // Name is given when registering, and can not be changed
    }

    public void setPetWeight(double newWeight) {
        weight = newWeight;
        // Weight changes every time you weigh your pet
    }

    public void setPetAge(double newAge) {
        age = newAge;
        // Surely age can change, too
    }
}
```

Summary: Constructor

- A special method with the same name as the class, and no return type
- Called only when an object is created
- It can take parameters to initialize instance variables
- You can define multiple constructors with different parameter lists

Next class (Tue, Nov 4)

- More on methods…
- Quiz on Tue