COMP 110
Introduction to Programming

Fall 2015
Time: TR 9:30 – 10:45
Room: AR 121 (Hanes Art Center)

Jay Aikat
aikat@cs.unc.edu

Previous Class

• What did we discuss?
Today

• Assignment 3 - extension:
  DUE Fri, 11/6 @ 11:55 PM

• Quiz on Tuesday

• Today – More on Constructors

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
Default Constructor

• Constructor that takes no parameters
  ```java
  public Pet()
  {
    name = "No name yet."
    age = 0;
    weight = 0;
  }
  ```

• Java automatically defines a default constructor if you do not define any constructors
  — You have not written a constructor explicitly, but you can still create objects

Constructor with Parameters

```java
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;
  }
}
```
A Closer Look

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

Same name as class name

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!

```java
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

```java
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

```java
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

```java
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 over time
  - If a value is never going to be changed, no setter is needed
Example: Initialize, Construct and Set

```java
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
    }
}
```

Static Variables

- Static variables are shared by all objects of a class
- Only one instance of the variable exists
- It can be accessed by all instances of the class

```java
public double gpa;
public static double highestGPA = 0.0;
public void setGPA(double newGPA) {
    if (newGPA > Student.highestGPA) {
        Student.highestGPA = newGPA;
    }
    gpa = newGPA;
}
```
Static Variables

- Static variables also called *class variables*
  - Contrast with *instance variables*
- Do not confuse class variables with variables of a class type
- Both static variables and instance variables are sometimes called *fields* or *data members*

Final Static Variables

- Variables declared *static final* are considered constants – value cannot be changed
  
  ```java
  public static final int MAX_CLASS_YEAR = 6;
  ```
- Now, this won’t work
  ```java
  public static void main(String[] args) {
      ...
      Student.MAX_CLASS_YEAR = 12;
      ...
  }
  ```
Static Methods

- Some methods may have no relation to any type of object
- Example
  - Compute max of two integers
  - Convert character from upper- to lower case
- Static method declared in a class
  - Can be invoked without using an object
  - Instead use the class name

The Math Class

- Provides many standard mathematical methods
  - Automatically provided, no import needed
- Example methods, figure 6.3a

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Example</th>
<th>Value Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>pow</td>
<td>Power</td>
<td>double</td>
<td>double</td>
<td>Math.pow(2.0, 3.0)</td>
<td>8.0</td>
</tr>
<tr>
<td>abs</td>
<td>Absolute value</td>
<td>int, long, float, or double</td>
<td>Same as the type of the argument</td>
<td>Math.abs(-7), Math.abs(7), Math.abs(-3.5)</td>
<td>7, 7, 3.5</td>
</tr>
<tr>
<td>max</td>
<td>Maximum</td>
<td>int, long, float, or double</td>
<td>Same as the type of the arguments</td>
<td>Math.max(5, 6), Math.max(5.5, 5.3)</td>
<td>6, 5.5</td>
</tr>
</tbody>
</table>
The Math Class

• Example methods

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Argument Type</th>
<th>Return Type</th>
<th>Example</th>
<th>Value Returned</th>
</tr>
</thead>
</table>
| min  | Minimum     | int, long, float, or double | Same as the type of the arguments | Math.min(5, 6)  
Math.min(5.5, 5.3) | 5  
5.3 |
| round | Rounding    | float or double | int or long, respectively | Math.round(6.2)  
Math.round(6.8) | 6  
7 |
| ceil | Ceiling     | double        | double      | Math.ceil(3.2)  
Math.ceil(3.9) | 4.0  
4.0 |
| floor | Floor       | double        | double      | Math.floor(3.2)  
Math.floor(3.9) | 3.0  
3.0 |
| sqrt | Square root | double        | double      | sqrt(4.0) | 2.0 |

Random Numbers

• `Math.random()` returns a random double that is greater than or equal to zero and less than 1
• Java also has a `Random` class to generate random numbers
• Can scale using addition and multiplication; the following simulates rolling a six sided die

```java
int die = (int) (6.0 * Math.random()) + 1;
```
Next class

- Quiz on calling methods from other methods, constructors, setters and getters