Constructors & Static Methods

COMP 110
Summer II 2012

Peter Lincoln
7/11/2012
Outline

- Constructors
- Static Methods
The Perils of Incorrectly Initialized Data
After heavy rains, water affected air-data sensors

These sensors feed angle of attack and yaw data to flight-control system

Water distorted preflight readings in 3 of the plane’s 24 sensors

Caused flight-control system to make an erroneous correction, making the plane stall and crash

http://youtu.be/_ZCp5h1gK2Q
Constructors

- Create and initialize new objects

- Special methods that are called when creating a new object

Examples

- Student jack = new Student();
- Book book = new Book("Java");
- Scanner keyboard = new Scanner(System.in);
Creating an Object

- Create an object jack of class Student
  - `Student jack = new Student();`

Assign memory address of object to variable
Allocate memory for an object and return that address
Create an object

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

- Can perform any action you write into a constructor’s definition

- Intended to perform initializing actions
  - For example, initializing the values of instance variables
Constructors: Similar to Mutators

- Mutators modify instance variables’ values

- Constructor *create* an object in addition to initializing it

- Like methods, constructors can have parameters
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet()
    {
        name = "Nameless";
        age = 0;
        weight = 0;
    }

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

- Calling a constructor with arguments is like calling a mutator method with arguments

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

No return type

Same name as class name

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

- Constructors give values to all instance variables

- Even if you do not explicitly give an instance variable a value in your constructor, Java will give it a default value
  - This is usually zero or null

- Normal programming practice is to give explicit values to all instance variables
Default Constructor

- A constructor that takes no parameters

- Pet Example
  - `public Pet() {
    name = "Nameless";
    age = 0;
    weight = 0;
  }

- Java automatically defines a default constructor if you do not define any constructors
Default Constructor

- If you define at least one constructor, a default constructor will *not* be automatically created for you.
Multiple Constructors

- You can have several constructors per class
  - They all have the same name
  - They each have different parameters
public class Pet
{
    private String name;
    private int age;
    private double weight;

    public Pet()
    {
        name = "Nameless";
        age = 0;
        weight = 0;
    }

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

- **Example:**
  ```java
  Pet myPet;
  myPet = new Pet("Frostillicus", 3, 121.5);
  ```

- **Cannot use an existing object to call a constructor:**
  ```java
  myPet.Pet("Fang", 3, 155.5); // invalid
  ```

- **Could use a mutator to edit an object:**
  ```java
  myPet.setPet("Fang", 3, 155.5);
  ```
Calling Methods from Constructors

- Just like calling methods from methods

```java
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;
}
```
Calling Methods from Constructors

- Can cause problems when calling public methods
  - Problem has to do with inheritance (Chapter 8)
  - Another class can alter the behavior of public methods

- Can solve this problem by making any method that a constructor calls private
public class Pet {
    private String name;
    private int age;
    private double weight;

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

    public void setPet(String newName, int newAge, double newWeight) {
        set(newName, newAge, newWeight);
    }

    private void set(String newName, int newAge, double newWeight) {
        name = newName;
        age = newAge;
        weight = newWeight;
    }
}
Syntax Summary

- Constructor Heading
  - AccessModifier ClassName(Parameter_List)
  - Where Parameter_List is
    - Empty (default constructor)
    - Type1 paramName1, Type2 paramName2, ...

- Creating Object
  - new ClassName(Argument_List)
  - Where Argument_List is
    - Empty (default constructor)
    - Expression1, Expression2, ...
Outline

- Constructors
- Static Methods
Instance Variables & Methods

- **Instance variables**
  - `private int age;`
  - `private String name;`

- **Methods**
  - `public int getAge()`
    ```
    {
      return age;
    }
    ```

- **Calling methods on objects**
  - `Student std = new Student();
    std.setAge(20);
    System.out.println(std.getAge());`
What about a `pow` method?

```java
public int simplePow(int base, int exp)
{
    // Returns base raised to the exp-th power,
    // where exp >= 0

    int result = 1;
    for (int i = 0; i < exp; i++)
    {
        result *= base;
    }
    return result;
}
```

- Do we really need an object to call this method?
Keyword: static

- Static variables and methods belong to a class as a whole, not to an individual object

- Where have we seen static before?
  - In the definition of named constant values

- When would you want a method that does not need an object to be called?
  - When the method has no relation to an object
static Examples

- Static Constants and Variables
  - `private static final int FACE_DIAMETER = 200;`
  - `public static final int FEET_PER_YARD = 3;`
  - `private static int numberOfInvocations;`

- Static Methods
  - `public static void main(String[] args)`
  - `public static int simplePow(int base, int exp)`
public static int simplePow(int base, int exp)
{
    // Returns base raised to the exp-th power, where exp >= 0
    int result = 1;
    for (int i = 0; i < exp; i++)
    {
        result *= base;
    }
    return result;
}
Accessing Static Members

- Static variables and methods can be accessed using the class name itself
  - DimensionConverter.FEET_PER_YARD
  - int z = Math.pow(2, 4);
  - double d = Double.parseDouble("5.05");
public class SomeClass
{
    public static final double PI = 3.14159;
    private boolean sunny = true;

    public static double area(double radius)
    {
        sunny = false;
        return PI * (radius * radius);
    }
}
Will this code compile?

```java
public class SomeClass {
    public static final double PI = 3.14159;
    private boolean sunny = true;

    public static double area(double radius) {
        sunny = false;
        return PI * (radius * radius);
    }
}
```

- Code will not compile
- Static methods are invoked without an object
  - No access to instance variables or non-static methods
  - No this
public class SomeClass
{
    public static final double PI = 3.14159;
    public int data = 12;

    private void printData()
    {
        System.out.println(data);
    }

    public static double area(double radius)
    {
        printData();
        return PI * (radius * radius);
    }
}
Will this code compile?

```java
public class SomeClass {
    public static final double PI = 3.14159;
    public int data = 12;

    private void printData() {
        System.out.println(data);
    }

    public static double area(double radius) {
        printData();
        return PI * (radius * radius);
    }
}
```

ERROR!
Will this code compile?

```java
public class SomeClass {
    public static final double PI = 3.14159;

    private void printPi() {
        System.out.println(PI);
        System.out.println(area(3.0));
    }

    public static double area(double radius) {
        return PI * (radius * radius);
    }
}
```
Will this code compile?

```
public class SomeClass {
    public static final double PI = 3.14159;
    private void printPi() {
        System.out.println(PI);
        System.out.println(area(3.0));
    }
    public static double area(double radius) {
        return PI * (radius * radius);
    }
}
```

- Nonstatic methods CAN call methods and access static variables

7/11/2012
Calling a Nonstatic Method from a Static Method

```java
public class SomeClass {
    public static final double PI = 3.14159;

    private void printPi() {
        System.out.println(PI);
        System.out.println(area(3.0));
    }

    public static double area(double radius) {
        SomeClass sc = new SomeClass();
        sc.printPi();
        return PI * (radius * radius);
    }
}
```
Self-Test Questions

- Can you call a nonstatic method from a static method?

- Can you call a static method from a nonstatic method?

- Can you access an instance variable inside a static method?
main is a static method

```java
import java.util.*;

public class MyClass
{
    public static void main(String[] args)
    {
        System.out.println("Give me a number, and I will "
                        + "tell you its square and its square’s square.");
        Scanner kb = new Scanner(System.in);
        int num = kb.nextInt();
        int numSquared = num * num;
        System.out.println("The square is " + numSquared);
        int numSquaredSquared = numSquared * numSquared;
        System.out.println("The square’s square is "
                        + numSquaredSquared);
    }
}
```
import java.util.*;

public class MyClass
{
    public static int square(int x)
    {
        return x * x;
    }

    public static void main(String[] args)
    {
        System.out.println("Give me a number, and I will "+"tell you its square and its square’s square.");
        Scanner kb = new Scanner(System.in);
        int num = kb.nextInt();
        int numSquared = square(num);
        System.out.println("The square is " + numSquared);
        int numSquaredSquared = square(numSquared);
        System.out.println("The square’s square is "
                           + numSquaredSquared);
    }
}
The Math Class

- Provides many standard mathematical methods, all static
  - Do not create an instance of the Math class to use its methods

- Call Math class’ methods using class name
  - Math.abs, Math.max, Math.min, Math.pow, Math.round, ...

- Predefined Constants
  - Math.PI, Math.E
Math.round

- Math.round: returns closest long (or int, if using a float) to argument
  - Math.round(2.3)
    - Returns 2
  - Math.round(2.7)
    - Returns 3
Math.floor

Math.floor: returns largest double value less than or equal to argument and equal to a mathematical integer

- Math.floor(2.3)
  - Returns 2.0
- Math.floor(2.7)
  - Returns 2.0
Math.ceil

- Math.ceil: returns smallest double value greater than or equal to argument and equal to a mathematical integer
  - Math.ceil(2.3)
    - Returns 3.0
  - Math.ceil(2.7)
    - Returns 3.0
What if you want an int?

- `Math.ceil` returns a double
  - `Math.ceil(5.6)` returns 6.0

```java
int num = (int)Math.ceil(5.6);
```
Wrapper Classes

- All primitive types have an associated wrapper class

- We have seen some of these before
  - Byte, Short, Integer, Long, Float, Double, Character, Boolean
Wrapper Classes

```java
int num = Integer.parseInt("36");
```

- `Integer.parseInt`, `Double.parseDouble`, etc. are all static methods.
- These wrapper classes also have some nonstatic methods.
Useful Static Constants & Methods

**Numeric**
- `Integer.MAX_VALUE`
- `Double.MIN_VALUE`
- `Float.parseFloat("23.7")`
- `Long.toString(368)`

**Character**
- `Character.toUpperCase`
- `Character.toLowerCase`
- `Character.isUpperCase`
- `Character.isLowerCase`
- `Character.isWhiteSpace`
- `Character.isLetter`
- `Character.isDigit`
Questions?
Logistics

- Next:
  - Lab 6