Constructor & Static Variables/Methods

Cheng, Wei  COMP110-001  June 3, 2014
Review of Pass-By-Value

• What is the output?

```java
public void swap(Student s1, Student s2) {
    Student s3 = s1;
    s1 = s2;
    s2 = s3;
}

Student berkeley = new Student();    berkeley.setYear(2);
Student brett = new Student();        brett.setYear(3);

swap(berkley, brett);
System.out.println(berkley.year);
```
Review of Pass-By-Value

• What is the output?

```java
public void swapYear(Student s1, Student s2) {
    int year = s1.year;
    s1.year = s2.year;
    s2.year = year;
}

Student berkeley = new Student();   berkeley.setYear(2);
Student brett = new Student();   brett.setYear(3);

swapYear(berkeley, brett);
System.out.println(berkeley.year);
```
Constructors

Recall our Student example:

```java
public class Student {
    private int PID;
    private int year;
    .... Accessors & mutators ..... 
}
```

Every time we use it, we have to use multiple statements to initialize values of instance variables.

```java
Student berkeley = new Student();
berkeley.setPID( 1234 );
berkley.setYear( 2 );
```
Constructors

• Isn’t this troublesome? What if we have some more complicated initialization steps? (e.g., an instance variable of class type to be initialized)

```java
Student berkeley = new Student();
berkeley.setPID(1234);
berkley.setyear(2);
....
```
Constructors

- Constructor is a special method that is called when a new object is created

```java
Student berkeley;  // not called

Student berkeley = new Student();  
// called with new keyword
```
Constructors

• Let’s write our first constructor:

```java
public class Student {
    private int PID;
    private int year;
    .... Accessors & mutators ..... 

    public Student( int PID, int year ) {
        this.PID = PID;
        this.year = year;
    }
}
```
Constructors

```java
public Student( int PID, int year ) {
    this.PID = PID;
    this.year = year;
}
```

- With this constructor, we can now do:

```java
Student berkeley = new Student( 1234, 2 );
```
Multiple Constructors

• You can have multiple constructors in one class. They all have the same name, just different parameters

```java
public class Student {
    
    ....

    public Student( int PID, int year ) {
        this.PID = PID;
        this.year = year;
    }

    public Student( int PID ) {
        this.PID = PID;
        this.year = 1; // default case – the 1st year
    }
}
```
Constructors

• Generally, constructor should contain all initialization logic
  – assign initial values based on input parameters
  – assign default initial values without input
  – reserve resource, prepare input/output stream
  – whatever other logic necessary (e.g., error checking)

• We will see more examples later.
Default Constructor

• What if you did not write any constructor?

```java
public class Student {
    private int PID;
    private int year;
    .... No constructor .....
}
Student berkeley = new Student();
```

Java gives each class a default constructor if you did not write any constructor. It assigns a default value to each instance variable.

- integer, double: 0
- String and other class-type variables: null
- boolean: false
Constructors

• If you define at least one constructor, a default constructor will *not* be created for you
Example: Pet class

```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;
    }
}
```
Calling a constructor

Pet myPet;
myPet = new Pet("Lightning", 3, 121.5);

• You cannot use an existing object to call a constructor:

myPet.Pet("Fang", 3, 155.5); // invalid!
Calling methods from constructors

• Just like calling methods from methods

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

private void setPet(String newName, int newAge, double newWeight)
{
    name = newName;
    age = newAge;
    weight = newWeight;
}
```
More Complicated Issues

For CS students:

• Constructor is used to create instance of class
  – Can constructor be private?

  – Can a constructor call another constructor?
Static Members

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

Sounds weird, doesn’t it?
  - static is against OO in some sense

- Where have we seen static before?
- When would you want a method that does not need an object to be called?
What about a pow method?

// Returns x raised to the yth power, where y >= 0.
public int pow(int x, int y)
{
    int result = 1;
    for (int i = 0; i < y; i++)
    {
        result *= x;
    }
    return result;
}

Do we need an object to call this method?
static version of pow method

```java
public class Math {
    public static double pi = 3.1415926;
    // Returns x raised to the yth power, where y >= 0.
    public static int pow(int x, int y)
    {
        int result = 1;
        for (int i = 0; i < y; i++)
        {
            result *= x;
        }
        return result;
    }
}
```
• Static variables and methods can be accessed using the class name itself:
  – System.out.println( Math.PI );
  – int z = Math.pow(2, 4);
Another Example

```java
public class MainClass {
    public static void main(String[] args) {
        ....
    }
}
```

MainClass is the entry-point of one application. main method is the entry-point of the class. It is executed before any instance is created. Thus it has to be static.
static vs non-static

• All static members are at class level. They are accessed without creating any instance.

• Thus, there is no “current object” in writing static methods.

• static methods has no access to instance variables or non-static methods ( since they belong to instances )
Will this code compile?

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

    public static double area(double radius) {
        calculated = 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
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;

    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 static methods and access static variables
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);
    }
}
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();

        System.out.println("The square is "+square(num));

        System.out.println("The square’s square is "+square(square(num)));
    }
}
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`
    - Others

- Predefined constants
  - `Math.PI`
  - `Math.E`
The Random class

• Why methods in random are not static?
• Don’t we just want a method that returns a random number whenever called?

Random rand = new Random();
int newNum = rand.nextInt();
Next Class

• Lab
• Bring your laptop & textbook
• Review slides