

COMP 110-001

Constructors and Static Methods

Yi Hong

June 03, 2015

Constructors

Recall our Student example:

```
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.

```
Student berkeley = new Student();  
berkeley.setPID( 1234 );  
berkeley.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)

```
Student berkeley = new Student();
```

```
berkeley.setPID(1234);
```

```
berkeley.setYear(2);
```

```
....
```

Constructors

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

```
Student berkeley; // not called
```

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

Constructors

- Let's write our first constructor

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

There is no return type or "void" keyword

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

Constructor has the same name as the class

Constructors

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

- With this constructor, we can now do:

```
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

```
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?

```
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

```
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); // x invalid!
```

Calling Methods from Constructors

- Just like calling methods from methods

```
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

- Constructor is used to create instance of class
 - Can constructor be private?
 - Can a constructor call another constructor?

```
public Pet(String initName)
{
    this(initName, 0, 0.0);
}
```

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

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?

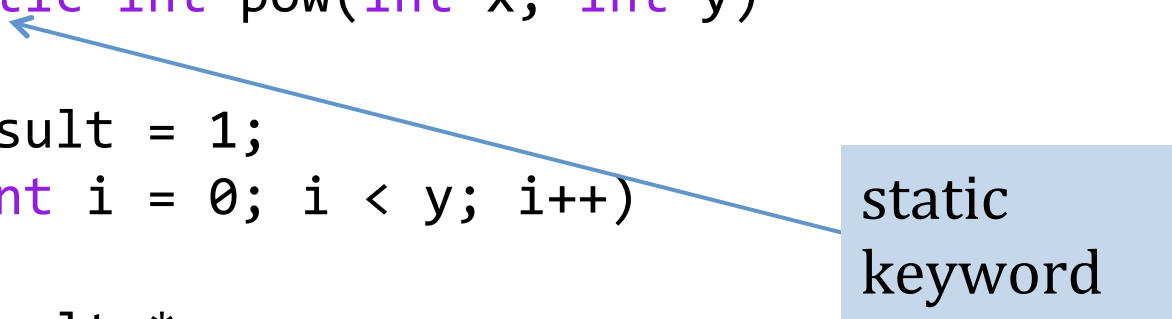
What about a pow method?

```
// 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

```
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
keyword

static

- 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

```
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?

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

    public static double area(double radius)
    {
        calculated = true;
        return PI * (radius * radius);
    }
}
```

ERROR!

- Code will not compile
- static methods are invoked without an object
 - no access to instance variables or non-static methods

Will This Code Compile?

```
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?

```
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);
    }
}
```

- Non-static methods CAN call static methods and access static variables

Calling a non-static method from a static method

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

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

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


main is a static method

```
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);
    }
}
```

static Helper Methods

```
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`, and others
- Predefined constants
 - `Math.PI`
 - `Math.E`

The Random class

- Why methods in random are not static?
- We want a method that returns a random (different) number whenever it is called.

```
Random rand = new Random();
```

```
int randNum = rand.nextInt();
```

Next Class

- Designing methods and overloading
- Package & review of classes