
CMPS 4143 - Topics in Contemporary Programming Languages

# L5- Java Class

Images and text taken from textbook unless otherwise noted.

# Definitions

- Object-oriented Programming
  - Classes
  - Objects
  - **Encapsulation**
  - **Inheritance**
  - **Polymorphism**
- Class vs Object (instance of class)

# More Definitions

- State - current set of values of the object
- Methods - operate on objects; may change state; state may affect behavior
- Inheritance in Java
  - a class ‘extends’ another class
  - has all the properties and methods of class extended and new methods and data fields that apply to new class

- Example: Math
  - only encapsulates methods, no data
  - do NOT construct an instance of class Math!
  - Call methods by using class name

```
Math.sqrt(x);
```

Example: Date

- construct them specifying initial state using new, then apply methods

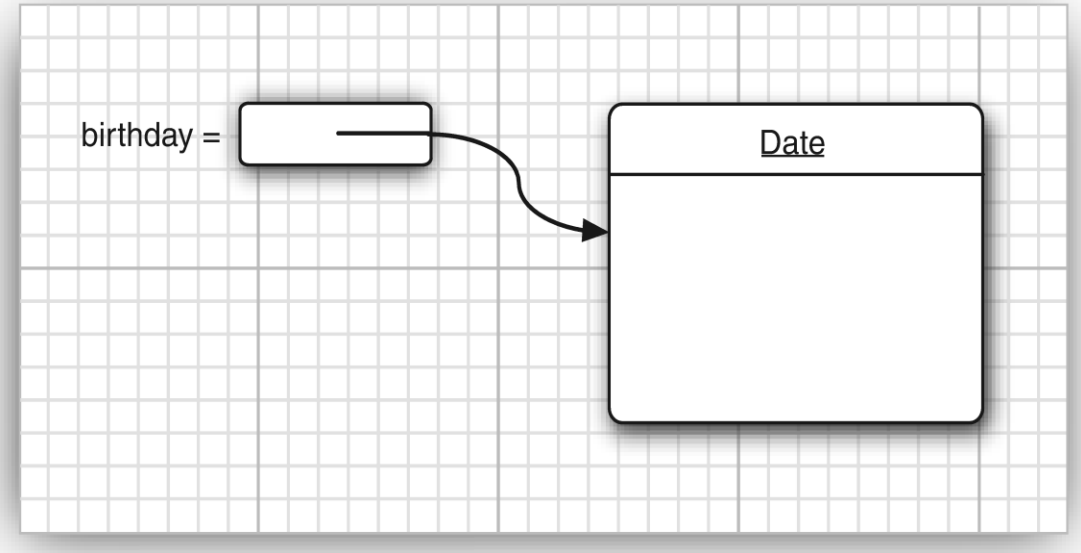
```
Date birthday = new Date();
```

# Using Existing Classes (..)

Difference between object and object variable!!!

```
Date birthday = new Date();
```

- `birthday` refers to an object



# Using Existing Classes (..)

```
Date deadline; //object variable
String s = deadline.toString(); //NO!
```

Need to point the object variable to an object first:

```
deadline =new Date(); // construct new object
```

or

```
deadline = birthday; // point to existing object
```

# Using Existing Classes (..)

## JAVA

```
Date birthday;
Date birthday = new Date();
```

- **Objects live on heaps, access bad reference, get error**
- **Auto garbage collection**
- **Clone to get a copy**

## C++

```
Date * birthday;
Date * birthday = new Date();
```

- **Access bad pointer, get another memory location**
- **Destructors necessary**
- **Effort for copy and assignment**

# Mutator and Accessor methods

- Mutator methods change state of object
- Accessor methods do NOT.
  - In C++ should use const suffix for these
  - no distinction in Java
- Convention:
  - mutators prefix method name with ***set***
  - accessors use prefix ***get***

# Building your own class

- A complete Java program requires one class with a main method.
- No other classes have a main method

# Simplest form of a class

```
class NameOfClass
{
        field1
        field2
        :
        constructor1
        constructor2
        :
        method1
        method2
        :
}
```

**no semicolon**

# A Dog Class

```
public class Dog {
    String breed;
    int age;
    String color;

    void barking() {
    }

    void hungry() {
    }

    void sleeping() {
    }
}
```

# Multiple objects

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

# Multiple Classes

```
public class Main {
  int x = 5;
}

Class Second{
  public static void main(String[] args) {
    Main myObj1 = new Main();
    System.out.println(myObj1.x);
  }
}
```

# Class Attributes

```
public class Main {
  int x = 5;
  final int x = 10;
}

public static void main(String[] args) {
    Main myObj1 = new Main();
    myObj1.x = 25; // x is now 25
// will generate an error: cannot assign a value to a
final variable
    System.out.println(myObj1.x);
}
```

# Class Methods

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```

# Class Methods (..)

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

# Constructors

A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes:

```
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}
```

# Class Variables

**Local** variables − Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

**Instance** variables − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

**Class** variables − Class variables are variables declared within a class, outside any method, with the static keyword

# Constructors

```
public class Puppy {
    public Puppy() {
    }

    public Puppy(String name) {
        // This constructor has one parameter,
   name.
    }
}
```

# Class and object

```
public class Puppy {
   public Puppy(String name) {
      // This constructor has one parameter, name.
      System.out.println("Passed Name is :" + name );
   }

   public static void main(String []args) {
      // Following statement would create an object myPuppy
      Puppy myPuppy = new Puppy( "tommy" );
   }
}
```

# Accessing Instance Variables and Methods

```
/* First create an object */
ObjectReference = new Constructor();

/* Now call a variable as follows */
ObjectReference.variableName;

/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

# Accessing Instance Variables and Methods(…)

```
public class Puppy {
   int puppyAge;

   public Puppy(String name) {
      // This constructor has one parameter, name.
      System.out.println("Name chosen is :" + name );
   }

   public void setAge( int age ) {
      puppyAge = age;
   }

   public int getAge( ) {
      System.out.println("Puppy's age is :" + puppyAge );
      return puppyAge;
   }
```

# Accessing Instance Variables and Methods(…)

```
public static void main(String []args) {
      /* Object creation */
      Puppy myPuppy = new Puppy( "tommy" );

      /* Call class method to set puppy's age */
      myPuppy.setAge( 2 );

      /* Call another class method to get puppy's age */
      myPuppy.getAge( );

      /* You can access instance variable as follows as well */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

```
Name chosen is :tommy
Puppy's age is :2
Variable Value :2
```

# A Simple Case Study

For our case study, we will be creating two classes. They are Employee and EmployeeTest.

First open notepad and add the following code. Remember this is the Employee class and the class is a public class. Now, save this source file with the name Employee.java.

The Employee class has four instance variables - name, age, designation and salary. The class has one explicitly defined constructor, which takes a parameter.

# In class

Employee Class:

name, age, designation and salary

EmployeeTest Class:

1. Create 2 objects ( 2employees)
    1. Constructors
2. Set and get all attributes for 2 objects
3. Print the attributes after setting the values

# Questions?