Module 1 - Lecture 9

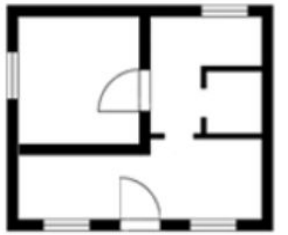
Classes & Encapsulation

Review

- Maps
- Sets
- Algorithmic complexity

Classes

A **class** is a grouping of variables and methods in a source code file from which we can generate objects.



Blueprint

Class

Object

Object

Constructor

- A special method called upon for initialization.

- No return type.
- Same name as the class.
- Can have 0 or many.

```
public class Lecture {
    public Lecture() {
        // initialize stuff here
    }
}
```

Methods

```
public class Lecture {
    public Lecture() {
        // initialize stuff here
    }

    public boolean isMethod() {
        return true;
    }
}
```

Fields / Properties / Attributes

```
public class Lecture {
    private int module;
    private int day;
    private String topic;

    public Lecture() {
        // initialize stuff here
    }
}
```

Derived Property / Field

```
public class Lecture {
    private int module;
    private int day;
    private String topic;

    public Lecture() {
        // initialize stuff here
    }

     // e.g. M1D2 - Variables and Data Types
     public String getlectureTitle() {
         return “M” + module + “D” + day + “ - “ + topic;
     }
}
```

Getters / Setters

```
public class Lecture {
   private int module;
   private int day;
   private String topic;

   public Lecture() {
      // initialize stuff here
   }

    public int getModule() {
        return module;
    }

    public void setModule(int m) {
        module = m;
    }
}
```

Using a class

```
public class Lecture {
    public Lecture() {
        // initialize stuff here
    }

    public boolean isMethod() {
        return true;
    }
}


Lecture myLecture = new Lecture();
if(myLecture.isMethod()) {
  //
}
```

Access Modifiers

- Access modifiers can be applied to instance methods and variables, as well as static methods and variables.
- They control whether certain methods and properties are available for use by the users of the class, or meant only to be used internally.

Overloading

You may provide flexibility to the user of your class by overloading methods or constructors.

- Overloaded methods must *have the same name*.
- Overloaded methods must *differ in the number of parameters, parameter types, or both*.
- Overloaded methods can have different return types, but that *must not* be the only difference.

Let's Code!

Static

- Belongs to the class
- Can only work with other static members/methods

Static Example 1 (No static)

```
public class Student {
  private String name;

  public Student(String name) {
    this.name = name;
  }

  public void setName(String newName) {
    name = newName;
  }
}
```

```
Student a = new Student("Walt");
a.setName("Bob");

Student b = new Student("Sam");
b.setName("Ashley");
```

a (instance)
~~name -> "Walt"~~
name -> "Bob"

```
void setName(String newName) {
  name = newName;
}
```

b (instance)
~~name -> "Sam"~~
name -> "Ashley"

```
void setName(String newName) {
  name = newName;
}
```

Student (class)
N/A

Static Example 2 (static attribute)

```
public class Student {
  private static String name;

  public Student(String name) {
    this.name = name;
  }

  public void setName(String newName) {
    name = newName;
  }
}
```

```
Student a = new Student("Walt");
a.setName("Bob");

Student b = new Student("Sam");
b.setName("Ashley");
```

a (instance)
name -> Student.name

```
void setName(String newName) {
  name = newName;
}
```

b (instance)
name -> Student.name

```
void setName(String newName) {
  name = newName;
}
```

Student (class)
~~name -> "Walt"~~
~~name -> "Bob"~~
~~name -> "Sam"~~
name -> "Ashley"

Static Example 2 (static method)

```
public class Student {
  private String name;

  public Student(String name) {
    this.name = name;
  }

  public static void setName(String newName) {
    name = newName;
  }
}
```

```
Student a = new Student("Walt");
a.setName("Bob");

Student b = new Student("Sam");
b.setName("Ashley");
```

COMPILER ERROR

Cannot make a static reference to the non-static field "name".

a (instance)
name -> ?

b (instance)
name -> ?

Student (class)

```
void setName(String newName) {
  name = newName;
}
```

Static Example 2 (everything static)

```
public class Student {
  private static String name;

  public Student(String name) {
    this.name = name;
  }

  public static void setName(String newName) {
    name = newName;
  }
}
```

```
Student a = new Student("Walt");
a.setName("Bob");

Student b = new Student("Sam");
b.setName("Ashley");
```

a (instance)
name -> Student.name

b (instance)
name -> Student.name

Student (class)
~~name -> "Walt"~~
~~name -> "Bob"~~
~~name -> "Sam"~~
name -> "Ashley"

```
void setName(String newName) {
  name = newName;
}
```

Let's Code!

Encapsulation

- The packaging of data and behavior into a single component.

- Hiding the implementation details of a class to prevent other parties from setting the data to an invalid or inconsistent state and also reduce coupling.

Goals of Encapsulation

- Code that is extendable
- Code that is maintainable
- Promote loose coupling

QUESTIONS?