RECITATION 3

Interfaces and Constructors

Why interfaces?

- We can define what we expect a class to do without writing any code
- Multiple classes that have similar functions can be made to follow certain conventions to allow for **polymorphism**
  - Polymorphism = Letting code correctly work with different class types

Example:

```
public interface PassFail {
    public boolean getPassing();
    public int getScore();
    public void setScore(int score);
}
```

Example:

```
public interface PassFail {
    public boolean getPassing();
    public int getScore();
    public void setScore(int score);
}
```

```
public class PassFailBean implements PassFail {
    int score;

    public void setScore(int newScore) {
        score = newScore;
    }
    public int getScore() {
        return score;
    }
    public boolean getPassing() {
        if(score<69) {
            return false;
        }
        return true;
    }
}
```

Notice that I use two variables in one class but only one in the other

```
public class AnotherPassFailBean implements PassFail{
    boolean passing;
    int score;

    public boolean getPassing() {
        return passing;
    }
    public int getScore() {
        return score;
    }
    public void setScore(int score) {
        this.score = score;
        if(score < 69) {
            this.passing = false;
        }else {
            this.passing = true;
        }
    }
}
```

Two classes doing the same thing but implemented in different ways

Example:

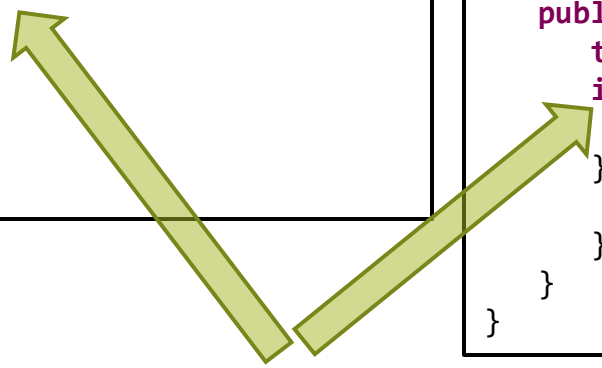
```
public interface PassFail {
    public boolean getPassing();
    public int getScore();
    public void setScore(int score);
}
```

```
public class PassFailBean implements PassFail {
    int score;

    public void setScore(int newScore) {
        score = newScore;
    }
    public int getScore() {
        return score;
    }
    public boolean getPassing() {
        if(score<69) {
            return false;
        }
        return true;
    }
}
```

```
public class LenientPassFailBean implements PassFail{
    boolean passing;
    int score;

    public boolean getPassing() {
        return passing;
    }
    public int getScore() {
        return score;
    }
    public void setScore(int score) {
        this.score = score;
        if(score < 50) {
            this.passing = false;
        }else {
            this.passing = true;
        }
    }
}
```



Two classes doing the DIFFERENT things but with the SAME interface

Example cont:

```
import bus.uigen.ObjectEditor;

public class Driver {
   public static void main(String[] args) {
      PassFail bean = new PassFailBean();
//    PassFail bean = new AnotherPassFailBean();
      bean.setScore(80);
      ObjectEditor.edit(bean);
   }
}
```

- What happens when I switch between the two instantiations of PassFail bean?

Constructors

- Constructors allow our program to do something upon being instantiated.
- Constructors are methods that do not have a return type (not even void) and have the same name as the class.
- If no constructor is called then the default constructor is automatically added in by java.

Example

- Take the code I had before and add a constructor:

```
public class PassFailBean implements PassFail {
   int score;

   public PassFailBean(int defaultScore) {
      setScore(defaultScore);
   }

   public void setScore(int newScore) {
      score = newScore;
   }
   public int getScore() {
      return score;
   }
   public boolean getPassing() {
      if(score<69) {
         return false;
      }
      return true;
   }
}
```

- Now when I create a new PassFailBean I can pass a starting score instead of calling the setScore method before the getScore or getPassing methods

Coding assignment:

- Adding on to what you did last week:
  - If you haven't already, create an Interface called UppercaseFilter with the following methods:
    - public String getInputString()
    - public void setInputString(String input)
    - public String getUppercaseLetters()
  - Create a new class called AnotherUppercaseFilter and have it implement UppercaseFilter.
  - Instead of having two global variables, this time only use one global variable
  - Change the line starting AnUppercaseFilter filter … to either:

```
UppercaseFilter filter = new AnUppercaseFilter();
```

  - OR (try switching between the two)

```
UppercaseFilter filter = new AnotherUppercaseFilter();
```

    - Does this change the functionality?
  - Try adding a constructor to AnUppercaseFilter that sets the inputString upon instantiation