Utilities

Creating reusable validation methods

Produced by: Dr. Siobhán Drohan
Ms. Mairéad Meagher
Ms. Siobhan Roche

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.setu.ie/

```java
package models;

public class Product {

   private String productName = "";
   private int productCode = -1;
   private double unitCost = 0;
   private boolean inCurrentProductLine = false;

   public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
      this.productName = productName;

      if ((productCode >= 1000) && (productCode <= 9999)){
         this.productCode = productCode;
      }

      this.unitCost = unitCost;
      this.inCurrentProductLine = inCurrentProductLine;
   }

   public int getProductCode() {
      return productCode;
   }

   public void setProductCode(int productCode) {
      if ((productCode >= 1000) && (productCode <= 9999)){ {
         this.productCode = productCode;
      }
   }
}
```

Validating Product Code

Must be between 1000 and 9999 inclusive.

When a Product is instantiated and the productCode is not matching those values, productCode should be -1.

```java
package models;

public class Product {

   private String productName = "";
   private int productCode = -1;
   private double unitCost = 0;
   private boolean inCurrentProductLine = false;

   public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
      this.productName = productName;

      if ((productCode >= 1000) && (productCode <= 9999)){
         this.productCode = productCode;
      }

      this.unitCost = unitCost;
      this.inCurrentProductLine = inCurrentProductLine;
   }

   public int getProductCode() {
      return productCode;
   }

   public void setProductCode(int productCode) {
      if ((productCode >= 1000) && (productCode <= 9999)){ {
         this.productCode = productCode;
      }
   }
}
```

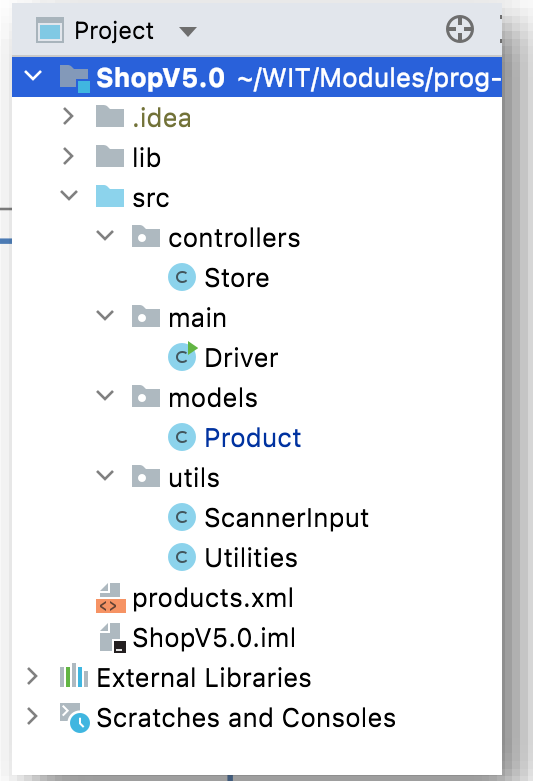
Validating Product Code

Repeated Code breaks the DRY principle. Don't Repeat Yourself!

Utility Classes and Helper Methods

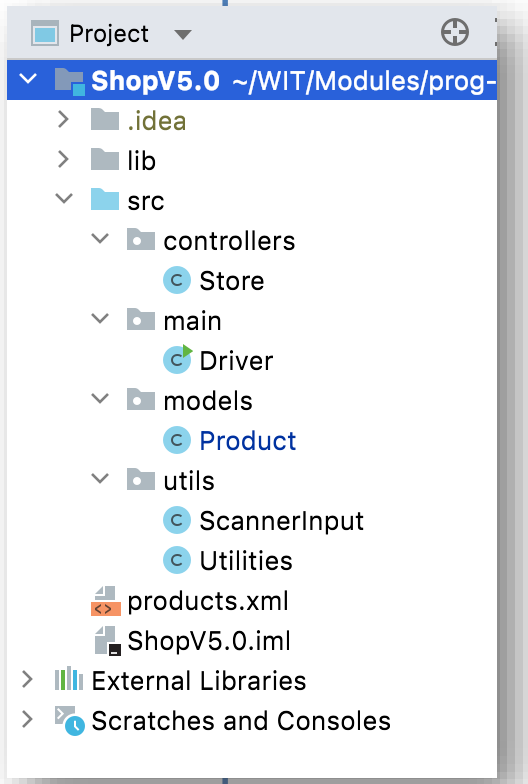
- To avoid repeated code like this, we can write a “helper” method that will perform this validation for us.
- We try to make these helper methods as RE-USABLE as possible.
- We generally store these RE-USABLE helper methods in Utility classes.
- These Utility classes can then be reused in any of our future projects.
- Any project can have any number of Utility classes e.g. ScannerInput is a Utility class!

Utilities Class




```
package utils;

public class Utilities {

   /**
    * This method returns true if the numberToCheck is between min and max (both inclusive)
    *
    * @param numberToCheck The number whose range is being checked.
    * @param min The minimum range number to check against (inclusive)
    * @param max The maximum range number to check against (inclusive)
    * @return Returns true if numberToCheck is between min and max (both inclusive), false otherwise.
    */
   public static boolean validRange(int numberToCheck, int min, int max) {
      return ((numberToCheck >= min) && (numberToCheck <= max));
   }

}
```

```java
package models;

import utils.Utilities;

public class Product {
    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        this.productName = productName;

        setProductCode(productCode);

        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public int getProductCode() {
        return productCode;
    }

    public void setProductCode(int productCode) {
        if (Utilities.validRange(productCode, 1000, 9999)) {
            this.productCode = productCode;
        }
    }
}
```

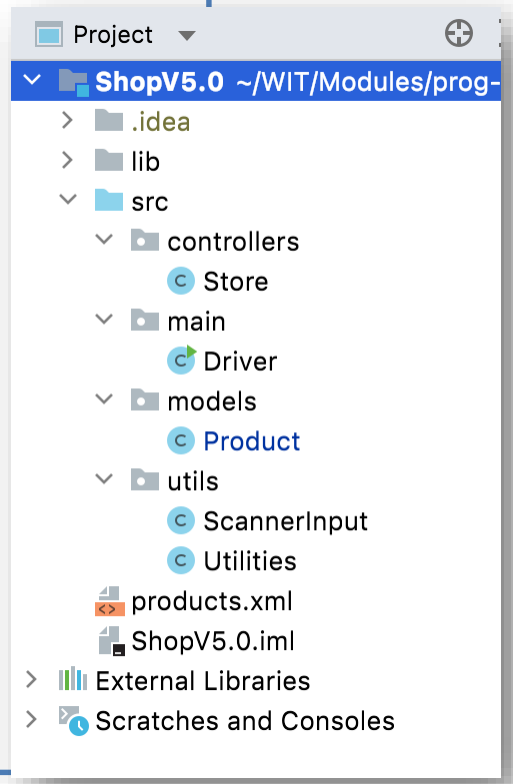


Using the Utility method: validRange in the Product class.

VALIDATING STRINGS USING UTILITIES

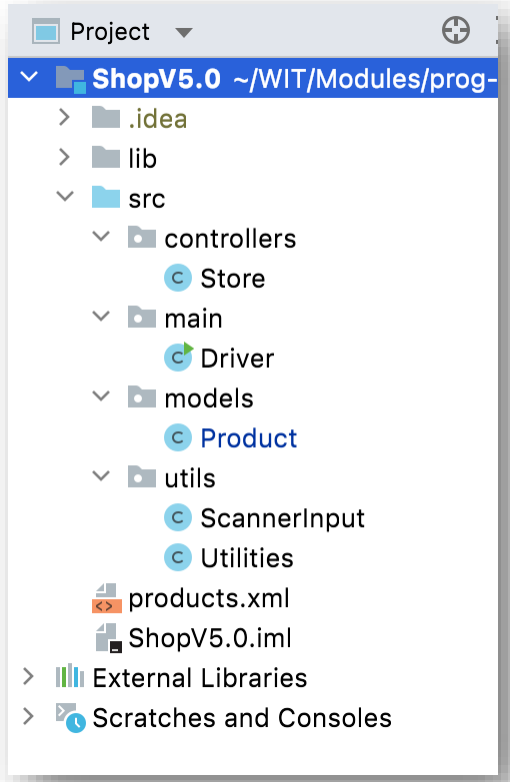
```java
public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {
        if (productName.length() <= 20){
            this.productName = productName;
        }else{
            productName.substring(0,20);
        }
        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public void setProductName(String productName) {
        if (productName.length() <= 20) {
            this.productName = productName;
        }
    }
}
```

Product Class – validate productName without utility methods (to max of 20 chars).



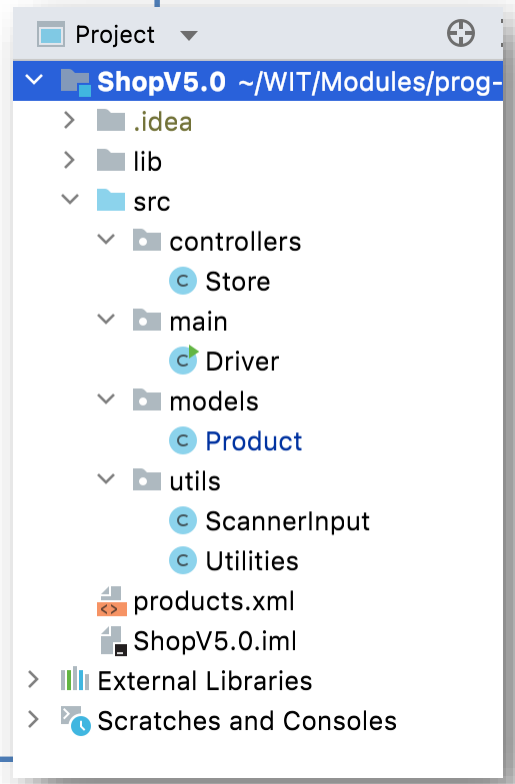
```java
/**
 * This method returns a string that was passed as a parameter, truncated to a specific length, also
 * passed as a parameter.  If the original String is less than the passed length, then the original string is just returned.
 *
 * @param stringToTruncate  The string that will be truncated to a specific length
 * @param length The length to which to truncate the string to
 * @return The string truncated to a specific length
 */
public static String truncateString(String stringToTruncate, int length){
    if (stringToTruncate.length() <= length) {
        return stringToTruncate;
    }
    else{
        return stringToTruncate.substring(0, length);
    }
}

/**
 * This method takes in a string, passed as a parameter and valdiates whether it is less than or equal to
 * a specific length or not.
 *
 * @param strToCheck The string that will be checked to see if it is a specific length
 * @param maxLength The length that the string will be validated against
 * @return True if the string is less than or equal the max length and false otherwise.
 */
public static boolean validateStringLength(String strToCheck, int maxLength){
    return strToCheck.length() <= maxLength;
}
```

Utilities Class – add two new methods



Product Class – using two new methods to validate productName.

```java
import utils.Utilities;

public class Product {

    private String productName = "";
    private int productCode = -1;
    private double unitCost = 0;
    private boolean inCurrentProductLine = false;

    public Product(String productName, int productCode, double unitCost, boolean inCurrentProductLine) {

        this.productName = Utilities.truncateString(productName, 20);

        setProductCode(productCode);
        this.unitCost = unitCost;
        this.inCurrentProductLine = inCurrentProductLine;
    }

    public void setProductName(String productName) {
        if (Utilities.validateStringLength(productName, 20)) {
            this.productName = productName;
        }
    }
```



Any Questions?