

## Homework 2

1. A T-shirt company makes shirts in 5 sizes:

<u>Size</u>	<u>Chest size (inches)</u>
Small	<= 32
Medium	33 - 36
Large	37 - 40
X Large	41 - 44
XX Large	45 - 48

When printing labels for T-shirts, the sizes are printed as S, M, L, XL, XXL.

Write a `Size` enum for these sizes. The `Size` enum should have 3 methods:

`toString` returns the size as letters for printing on labels: S, M, L, XL, XXL.

`intValue` returns the *maximum* chest circumference that can fit this size. For example: `Size.Medium.intValue()` returns 36.

`getSize( int )` is a **static** method that returns the correct shirt `Size` for a person's chest size. For example, `getSize(37)` returns `Size.Large`. Write a loop using the `enum values()` method to find the best size.

Don't use a `switch` statement or `if ... else if ... else if` in any of these methods. All the methods should work without modification if the store owner decides to change the inch value for some size, or if he adds a new size like XXX Large.

Submit a UML diagram and source listing for your `Size` enum. You may write by hand or print-out.

2. Consider this bit of code:

```
public class Person {
    private Date birthday;
    private String name;
    public Person( String name, Date birth ) { this.name = name; this.birthday = birth; }
    public Date getBirthday() { return birthday; }
    public String toString() { return name; }
}
```

And these 4 methods that use a `Person` reference:

```
public void a(Person p) { System.out.println( p ); }
public void b(Person p) { System.out.println( p.toString() ); }
public void c(Person p) { System.out.println( p.getBirthday() ); }
public void d(Person p) { System.out.println( p.getBirthday().getYear() + 1900 ); }
```

When each of these methods is invoked, some values of `p` and its attributes could cause **`NullPointerException`** to be thrown.

For each these methods, in what cases will a **`NullPointerException`** be thrown? For each method, list all possible cases (`p = null`, `p.name = null`, or `p.birthday = null`) that would throw a **`NullPointerException`**, but assume that `System.out` is *not* null.

3. Give a Java code example (1 or 2 statements for each) that throws each of these exceptions:

a) `ArithmeticException`

b) `ClassCastException`

c) `IllegalFormatException`

d) All the above are *unchecked exceptions*, meaning that we don't have to use try - catch in code that might throw them. Choose any one of the above exceptions and explain why it makes sense that it should be an *unchecked exception*.

**Note:** Another important exception is `ArrayIndexOutOfBoundsException`

4. A really useful exception is `IllegalArgumentException`. Give an example of a method in the `Purse` where it would make sense to throw this exception, and write some Java code for the start of the method showing when and how you would throw this exception.

`Greenfoot` uses this exception a lot. For example, if you call `setColor( "chocolate" )` and there's no color named chocolate.

5. We have a `Calculator` application what is called from a graphical UI. When the user presses a function key (like `sqrt`, `sine`, or `square`) is calls `Calculator` code like this:

```
public class Calculator {
    private double result; // last value remembered by calculator

    /** Perform an operation on the result stored inside calculator.
     *  @param op is the name of operation to perform.
     */
    public void perform(String op) {
        if ( op.equals("sine") )
            result = Math.sine( result );
        else if ( op.equals("sqrt") )
            result = Math.sqrt( result );
        else if ( op.equals("square") )
            result = Math.pow( result, 2 );
        else ...// more operations
    }
}
```

For example, to take square root of the calculator result, we would write:

```
calculator.perform( "sqrt" );
```

Using Strings and "if" - "else if" isn't a very good design. It would be more flexible and OO-style to use an *object* to represent the operation you want the calculator to perform.

Write example Java code (without comments, for a change) to show how you would do this:

a) Define an interface name `Function` that has one method named `perform` that defines the interface for operation like `Math.sine()`, `Math.sqrt`, or `Math.pow( ,2)`.

b) Define `Sqrt` as a class that implements `Function` and performs square root.

c) Rewrite the `Calculator perform` method so it can perform any `Function`, and without any if ... else if testing.

d) Draw a UML class diagram to summarize the design.