Q1: Given the following class:

```
public class Q1 {
        public int var1;
        public int var2;
        System.out.println("Exercise 2.r");

        public Q1() {
        }
        public Q1(int a , int b) {
                var1 = b;
                var2 = a;
        }
        public static void main(String args[]) {
                Q1 exercise2R = new Q1 (4,7);
                System.out.println(exercise2R.var1);
                System.out.println(exercise2R.var2);
        }
}
```

1. This program cannot be run.
2. [✓ This program does not compile.]
3. It will print 74.
4. It will print 47.

Q2: Consider this simple class representing a car and the level of fuel in the tank:

```
public class Car {
    private final String registration;
    private final double efficiency;  // fuel efficiency in litres/km
    private final double capacity;    // fuel capacity in litres
    private double fuel;              // current fuel level in litres
    public Car(String reg, double eff, double cap) {      registration = 
reg;
      efficiency = eff; // in litres/km
      capacity = cap;   // in litres
      fuel = 0.0;
    }
    public String getRegistration() {
      return registration;
    }
    public double getFuel() {
      return fuel;
    }
    public double getRange() {
      // calculate current range based on the efficiency and
      // current fuel level
      return fuel / efficiency;
    }
    public void addFuel(double moreFuel) {
      // Add more fuel
      this.fuel += moreFuel;
    }
    public void drive(double distance)  {
      // drive (and use some fuel)
      double usedFuel = efficiency * distance;
      fuel -= usedFuel;
    }
    public String toString() {
      return String.format("%s range: %6.2f km", registration,
getRange());
    }
  }
```

Here is an exception class intended to be used with the Car class:

```
public class FuelFillException extends Exception {
    public FuelFillException() {
      super();
    }
    public FuelFillException(String msg) {
      super(msg);
    }
  }
```

Modify Car's addFuel method so that it throws a FuelFillException as follows:

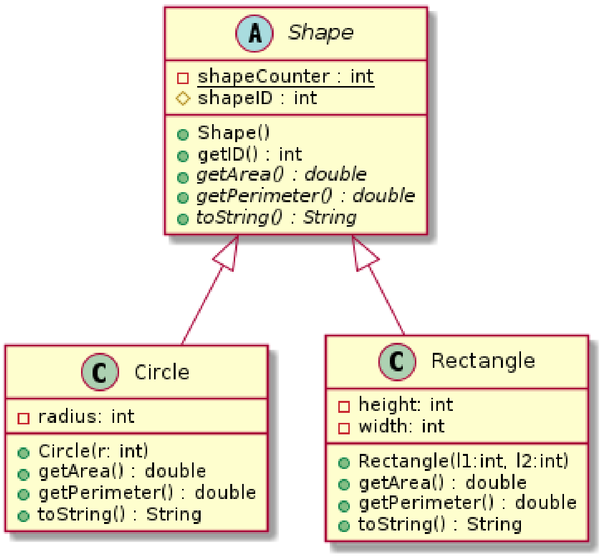
- if the amount of fuel to be added is less than or equal to zero; or
- if the amount of fuel to be added would overfill the tank beyond its capacity.

In each case, a suitable message should be included in the thrown exception. You answer should just be the new improved version of the addFuel method.

Answer:

```
public boolean addFuel(double moreFuel) throws FuelFillException {
    // Add more fuel
    // throw an exception if capacity would be exceeded
    if (moreFuel<=0.0)
      throw new FuelFillException(String.format("Amount of fuel (%4.2f)
must be positive", moreFuel));
    if (fuel + moreFuel > capacity)
      throw new FuelFillException(String.format("Fuel over-fill: %4.2fl
> %4.2fl", moreFuel, capacity-fuel));
    this.fuel += moreFuel;
  }
```

Q3: The classes are intended to be used for representing shapes used in a diagram:



There could multiple other different shapes - but only Circle and Rectangle are represented - each of which has different characteristics.

But all shapes have area and perimiter as properties that can be calculated.

Suppose that the classes represented in the UML diagram have been implemented in Java. Consider this program that makes use of them:

```
public class ShapeQuestion {

        public static void main(String[] args) {
            Circle c1 = new Circle(2);
            Circle c2 = new Circle(3);
            Rectangle r1 = new Rectangle(2,3);
            Rectangle r2 = new Rectangle(3,1);

            System.out.printf("%s%n%s%n%s%n%s%n",c1,c2,r1,r2);

            Object[] shapes = {c1,c2,r1,r2};
            for (Object o: shapes) {
              System.out.println(o);
            }

            double totalArea=0.0;
            for (Object o: shapes) {
              totalArea += o.getArea();
            }
            System.out.printf("Total area of the shapes is:
%4.2f%n",totalArea);

        }

}
```

Explain why the first for loop (that prints out the shapes from the shapes array) would work, but the second for loop (that is supposed to calculate the total area of the shapes) will not compile.

Suggest how you would change the code so that the second loop would work.

Answer:

The first loop uses the fact that all Shapes override the toString method from Object - so dynamic dispatch will be used on each object to select the correct toString method. ()

But in the second loop, the apparent type of o is Object, which does not have a getArea() method available. To fix, you could change the array type to Shape[] instead of Object[] throughout (and the array variables too). Alternatively you could cast each o in the second loop to Activity type:

```
totalArea += ((Shape) o).getArea();
```

Q4: Given the following hierarchy:

```
interface A {
        void method();
}

interface B implements A {
        static void staticMethod() {}
}

final class C implements B {
}

public abstract class D implements A {
        @Override
        void method() {
        }
}
```

Choose all the true statements:

1. Interface C does not inherit the staticMethod() method.
2. Interface B cannot implement another interface.
3. The class C implementing B also inherits the abstract method() of the A interface, and since it is not declared abstract it cannot be compiled.
4. Class D cannot be compiled because it cannot be declared abstract. In fact, it does not declare any abstract method.

5. Class D does not compile because the method it declares is not declared public.
6. Class D compiles only because the method is annotated with Override.

Q5: Given the following hierarchy:

```
public interface Technology {
        void print();
}

public class Inkjet implements Technology {
        @Override
        public void print() {
                System.out.println("Inkjet print");
        }
}

public class Laser implements Technology {
        @Override
        public void print() {
                System.out.println("Laser print");
        }
}
```

Create a generic Printer class parameterized with a technology, which in turn declares a print() method, which however delegates to its own technology the actual message to be printed.

Answer:

```
public class Printer<T extends Technology> {
private T technology;

public Printer(T technology) {

this.technology = technology;
}

public void print() {

technology.print();
}
}
```