Solution Serie 9 - Objects and Types

1 Theoretical Questions (6 points)

1. What is the difference between subtyping and subclassing? Provide an example for your explanation.

   Answer:

   Substitution (subtype) a specification notion. B is a subtype of A if an object of B can masquerade as an object of A in any context.

   Inheritance (subclass) should not be confused with subtyping. In general, subtyping establishes an is-a relationship, whereas subclassing only reuses implementation and establishes a syntactic relationship, not necessarily a semantic relationship (inheritance does not ensure behavioral subtyping).

   To distinguish these concepts, subtyping is also known as interface inheritance, whereas subclassing is known as implementation inheritance or code inheritance. Subclassing example is in question number 4.

2. Using the Java class-interface hierarchy given in Figure 1, explain what is the relationship between classes and interfaces.

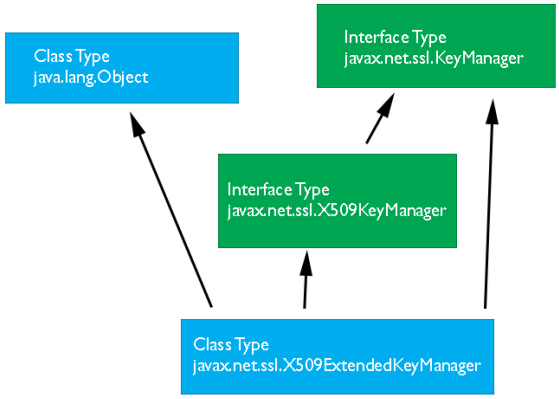

Figure 1: Java interface hierarchy

Answer:

`javax.net.ssl.X509ExtendedKeyManager` *is a subtype of* `java.lang.Object`.

`javax.net.ssl.X509ExtendedKeyManager` *is a subtype of* `javax.net.ssl.X509KeyManager`.

`javax.net.ssl.X509ExtendedKeyManager` *is a subtype of* `javax.net.ssl.KeyManager`.

`javax.net.ssl.KeyManager` *is a supertype of* `javax.net.ssl.X509ExtendedKeyManager`.

`javax.net.ssl.KeyManager` *is a supertype of* `javax.net.ssl.X509KeyManager`.

`java.lang.Object` is a supertype of all other class and interface types.

3. Which forms of polymorphism are used in the Java code in Listing 1? Explain each of the forms.

```
public class Bern<TT> { // Hint 2
    private TT var1;
    public void set(TT mh) { this.var1 = mh; }
    public TT get() { return var1; }

 public static void main(String[] args) {
   int a = 3;
   float b = 2F;
   b = a;   // Hint 1
   System.out.println(b);
   Bern<Integer> mj = new Bern();
   mj.set(12);
   System.out.println(mj.get());
   }
}
```

Listing 1: Forms of polymorphism

Answer:

Hint 1: Coercion
Hint 2: Parametric Polymorphism

4. Use Java subclassing to better structure the code in Listing 2 and avoid code cloning.

Answer:

```
class Bicycle {
private int frame_size;
public float price() { return 100 * 2; }
public float salesTax() { return price() * .08; }
}
class RacingBicycle extends Bicycle {
private int pieces_count;
public void calculateWeight();

}
```

5. In the code in q5.zip explain how covariant and contravariant are used in each code block. Why are there compile-time errors if you try to run the code?

Answer:

```
   class Bicycle {
      private int frame_size;

      // the price of this bicycle
      public float price() { return 100 * 2; }

      // the sales tax on this bicycle
      public float salesTax() { return price() * .08; }
}

   class RacingBicycle {
      private int frame_size;
      private int pieces_count;

      // the price of this bicycle
      public float price() { return 100 * 2; }

      // the sales tax on this bicycle
      public float salesTax() { return price() * .08; }

      // returns the weight of this bicycle
      public void calculateWeight();
}
```

Listing 2: Subclassing

Block 1: covariance
Block 2: contravariance
Block 3: contravariance
The <? extends Number> says the compiler uses some unknown (the ?) subtype of Number.
It explicitly constrains the wildcard (the ?) to represent the unknown subtype of Number.
In other words, the method is a covariant use of the List of Numbers; it works with any List of any subtype of Number.
The <? super Number> represents some unknown supertype of Number.
It is a contravariant use of the List of Numbers and it works for any List of any supertype of Number.
Truck is not a subtype of Car. Car is not a subtype of Truck.
The 4.2 number would not be allowed because the compiler cannot determine what the actual type of the object is in the generic structure.