Zwei Teile: Praktische und theoretische.

I. Das praktische Teil bestand aus drei Aufgaben: (insgesamt 30 P)

1. Eine neue Klasse + eine neue Funktion schreiben (insgesamt 15P)

1.a. Eine neue Klasse Bike schreiben. Ist ähnlich zu Car, nur der Rabatt etwas andres gerechnet wird. Da muss man noch sein Programm anpassen, sodass add und count mit einem Bus funktionieren.

1.b. Eine neue Funktion entwickeln, die alle Fahrzeuge ausgibt, die zwischen zwei Jahren gebaut wurden (zwischen min und max)

2. Eine neue Klasse TrafficController2LeftCars schreiben, die sich auf TafficControllerSimple basiert und bei der von rechts immer noch nur ein Fahrzeug auf der Brücke sein darf, aber von links dann zwei. (insgesamt 9 P)

3. Lambda Expressions an bestimmte Stellen einfügen. Es gab insgesamt drei solche Stellen und für den ersten richtigen Ausdruck hat man einen, für den zweiten zwei und für den dritten drei Punkte bekommen können. (insgesamt 6 P)

Es gab dafür drei vorgegebene Klassen: Person, Student und Worker. (Person{...int getAge() {return age;}}Student extends Person{...}, Worker extends Person{ … int getSalary(){return salary;} …})

- Es gab eine Liste<Person> mit verschiedenen Objekten(Personen, Students und Workers)
- Beim ersten Ausdruck mussten alle Personen aus der Liste ausgegeben werden, deren Älter > 25 ist.
- Beim zweiten alle Students, deren Älter < 30 ist.
- Beim letzten alle Workers, deren Gehalt > 2000 ist.

II. Bei dem theoretischen Teil gab es insgesamt 5 Fragen: (insgesamt 20P)

1.

(2 Points)

```
public class A {
   public static void sm() {System.out.print("1");}
   public void im() {System.out.print("2");}
}
public class B extends A {
   public static void sm() { System.out.print("3"); }
   public void im() { System.out.print("4"); }

   public static void main(String[] args) {
      B b = new B();
      A a = b;
      a.sm();
      a.im();
   }
}
```

What is the output of this program.

(Select one of following options:)

- a. 14
- b. 12
- c. 34 ✖
- d. 32

Your answer is incorrect.

2.

(7 Points)

Demonstrate how the direct dependency of class **Application** from class **MemoryOptimizedServer** can be removed by means of *Dependency Injection/Constructor Injection*.

Dependency Injection shall be accomplished by class **SetupApplication**.

Further demonstrate how for **app1** the class **MemoryOptimizedServer** and for **app2** the class **SpeedOptimizedServer** can be used. Accordingly modify/extend the code at the marked places.

Code changes are required at positions //1 - //7.

```
                                                          // 1

class Application {
     MemoryOptimizedServer srv = new MemoryOptimizedServer();   // 2
                                                                // 3
}

class MemoryOptimizedServer                                     // 4
{
   public String getServerType() { return "MemoryOptimized"; }
}

class SpeedOptimizedServer                                      // 5
{
   public String getServerType() { return "SpeedOptimized"; }
}

public class SetupApplication {
   public static void main(String[] args) {

      Application app1 = new Application();                     // 6
      Application app2 = new Application();                     // 7

       System.out.println("Server app1: " + app1.srv.getServerType();
       System.out.println("Server app2: " + app2.srv.getServerType();
      }
}
```

```
interface OptimizedServer{
public String getServerType();
}


class Application{
private OptimizedServer srv;
Application(OptimizedServer optimizedServer){
this.optimizedServer = optimizedServer
}
}


class MemoryOptimizedServer implements OptimizedServer{
...
}
class SpeedOptimizedServer implements OptimizedServer{
...
}
public class  SetupApplication{
public static void main (String [] args) {
Application app1 = new Application(new MemoryOptimizedServer);
Application app2 = new Application(new SpeedOptimizedServer);
...
}
}
```

3.

(2 Points)

```
class Circle {
      int x,y,r;
}

public class ParameterPassing {
   static void setRadius(int r, Circle c) {
      c.r = r; r = 0; c = null;
   }

   public static void main(String[] args) {
      int r = 5;
      Circle c = new Circle();
      c.r = 1;

      setRadius(r, c);

      System.out.println("r = " + r);
      System.out.println("c.r = " + c.r);
   }
}
```

What is the output of this program.

r = 5 ✔

c.r = 5 ✔

4.

(6 Points)

Assume the following Java classes and declarations.

```
class X {}
class Y extends X {}

X[] ax = new X[10];
Y[] ay = new Y[20];

List<X> lx = new ArrayList<X>();
List<Y> ly = new ArrayList<Y>();
List<?> lw;
List<Object> lo;

List<? super X> lsX;
List<? super Y> lsY;
```

Determine for each of the following assignments if it is correct or not.

lo = lw; Not Correct ✔

lx = ly; Not Correct ✔

ax = ay; Correct ✔

lw = lsY; Correct ✔

lsY = lsX; Not Correct ✖

lsX = ly; Not Correct ✔

5.

(3 Points)

```
public class Incrementer extends Thread {
      private static int counter = 0;
      private final int N;

      public Incrementer(int N) { this.N = N; }

      public static int getCounter() { return counter; }

      public void run() {
            for (int i=0; i < N; i++) increment();
      }

      private void increment() { counter++; }
}
```

Select below those variants of method **increment()** that ensure correct synchronization when **counter** is concurrently incremented by multiple instances of class **Incrementer**.

☐ a. **private synchronized void increment() { counter++; }**

☑ b. **private static synchronized void increment() { counter++; }** ✔

☐ c. **private void increment() {**
synchronized (getClass()) { counter++; }
}

☑ d. **private void increment() {** ✖
synchronized (this) { counter++; }
}