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;} …})

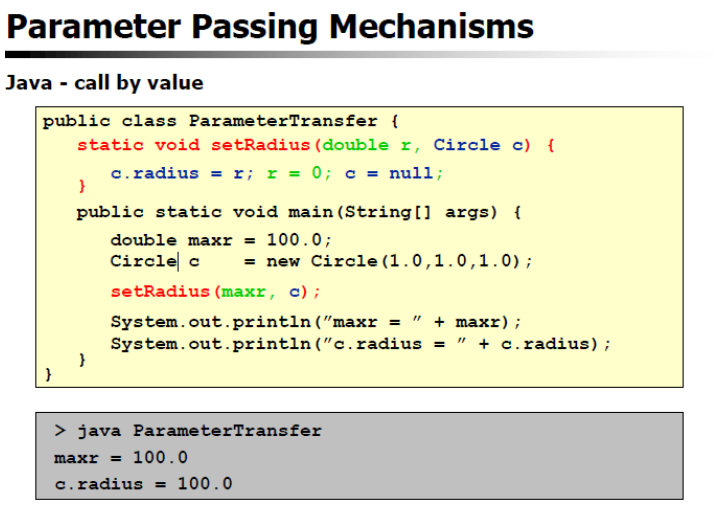
- Es gab eine Liste<Person> mit verschiedenen Objekten(Personen, Students und Workers)
- Beim ersten Ausdruck mussten alle Personen aus der Liste ausgegeben werden, deren Älter > 25 ist.
- Beim zweiten alle Students, deren Älter < 30 ist.
- Beim letzten alle Workers, deren Gehalt > 2000 ist.

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

1. Zwei Klassen A und B gegeben, B extends A. A hat zwei Methoden, die jeweils eine Zahl ausgeben und B hatte dieselben Methoden, nur es wurden andere Zahlen ausgegeben. Zu antworten war, was ca. so ein Programm ausgibt: A a = new A(); B b = new B(); a = b; b.methode1(); b.methode2();

2. Dependency Injection. Man musste dort direkte Dependency durch Dependency Injection ersetzen. (einfach genau so, wie es in VehicleManagement implementiert werden sollte)

3. Parameter Passing. Da gabs ca. so ein Beispiel:



Parameter Passing Mechanisms

Java - call by value

```
public class ParameterTransfer {
   static void setRadius(double r, Circle c) {
      c.radius = r; r = 0; c = null;
   }
   public static void main(String[] args) {
      double maxr = 100.0;
      Circle c    = new Circle(1.0,1.0,1.0);
      setRadius(maxr, c);
      System.out.println("maxr = " + maxr);
      System.out.println("c.radius = " + c.radius);
   }
}
```

```
> java ParameterTransfer
maxr = 100.0
c.radius = 100.0
```

4. Assignment Compatibility. Ca. so was in die Richtung:

Example: Assignment Compatibility

```
List<Person> lp;
List<Student> ls;
List<?> lw;
List l;
List<Object> lo;

lw = lp;
lw = ls;
lw = l;
lw = lo;

ls = lw;    // error
ls = lp;    // error
ls = l;     // unchecked
ls = lo;    // error

...
```

```
...

lp = lw;    // error
lp = ls;    // error
lp = l;     // unchecked
lp = lo;

l = lw;
l = lp;
l = ls;
l = lo;

lo = ls;    // error
lo = lp;    // error
lo = lw;    // error
lo = l;     // unchecked
```

Generics – Bounded Wildcards

- **Upper bounded wildcard:**

  `<? extends T>`

  Matches all types that are sub-types of T (including T)

- **Lower bounded wildcard:**

  `<? super T>`

  Matches all types that are super-types of T (including T)

Wildcards Guidelines - Example

```
List<? super Person> lSp1 = new ArrayList<Person>();
List<? super Person> lSp2 = new ArrayList<Student>();     // Error
List<? super Person> lSp3 = new ArrayList<Object>();

List<? extends Person> lEp1 = new ArrayList<Person>();
List<? extends Person> lEp2 = new ArrayList<Student>();
List<? extends Person> lEp3 = new ArrayList<Object>();    // Error
```

Generics – Subtyping

- Type parameters in Java are invariant, rather than covariant like arrays.

```
Person[] ap = new Person[100];
Student[] as = new Student[100];

ap = as;        // okay - arrays are covariant


List<Person> lp = new ArrayList<Person>();
List<Student> ls = new ArrayList<Student>();

lp = ls;        // Type mismatch: cannot convert from
                // List<Student> to List<Person>
```

5. Synchronisation. Man musste zwischen zwei Optionen wählen, wie das Programm richtig synchronisiert werden soll:

- Die Methode synchronisieren
- Die methode synchronisieren und static machen
- Das Statement drinnen synchronisieren
- Irgendwas, kann mich leider nicht mehr daran erinnern

Also ca. sowas:

Example – Synchronized Static Method

```
public class ThreadSync extends Thread {
   static int cnt = 0;
   public static void main(String[] args) throws
                                        InterruptedException {
      Thread t1 = new ThreadSync();
      Thread t2 = new ThreadSync();
      t1.start(); t2.start();
      t1.join(); t2.join();
      System.out.println(cnt);
   }
   public void run() {
      for (int i=0; i < 1000; i++){
                increment();
      }
   }

   private synchronized static void increment() { cnt++; }
}
```

Since `increment()` now is a static method, the thread acquires the intrinsic lock for the `class` object associated with the class.

```
Output: 2000
```