Excursus Interfaces

Interface

```java
package schnittstelle;

public interface IGraphicsCard {

    void display(String text);

}
```

Class implementing the interface (and thereby implementing the methods (here: display):

```java
package schnittstelle;

public class NVidia implements IGraphicsCard {

    public void display(String text) {
        System.out.println("NVidia is displaying the text " + text);
    }

}
```

Interfaces are a precondition for component oriented programming (weakly coupled components).

Interfaces represent "contracts", that other classes might sign by using the "implements" keyword.

Student.java StudentLocal.java HumanBeing.java

```java
package schnittstelle;

public class Motherboard {
    private IGraphicsCard gc;

    public Motherboard(IGraphicsCard gc) {
        this.gc = gc;
    }

    public void renderText (String text) {
        gc.display(text);
    }

}
```

NVidia.java

```java
package schnittstelle;

public class NVidia implements IGraphicsCard {

    public void display(String text) {
        System.out.println("NVidia is displaying the text " + text);
    }

}
```

Motherboard.java StartPc.java AMD.java IGraphicsCard.java

```java
package schnittstelle;

public interface IGraphicsCard {

    void display(String text);

}
```

Using the interface we were able to declare the class motherboard completly independently from a specific implementation of IGraphicsCard such as NVidia or AMD.

This type of programming is called "dependency injection" (the dependency on a specfic class e.g. AMD is "injected" via the constructor)

Motherboard.java StartPc.java AMD.java IGraphicsCard.ja

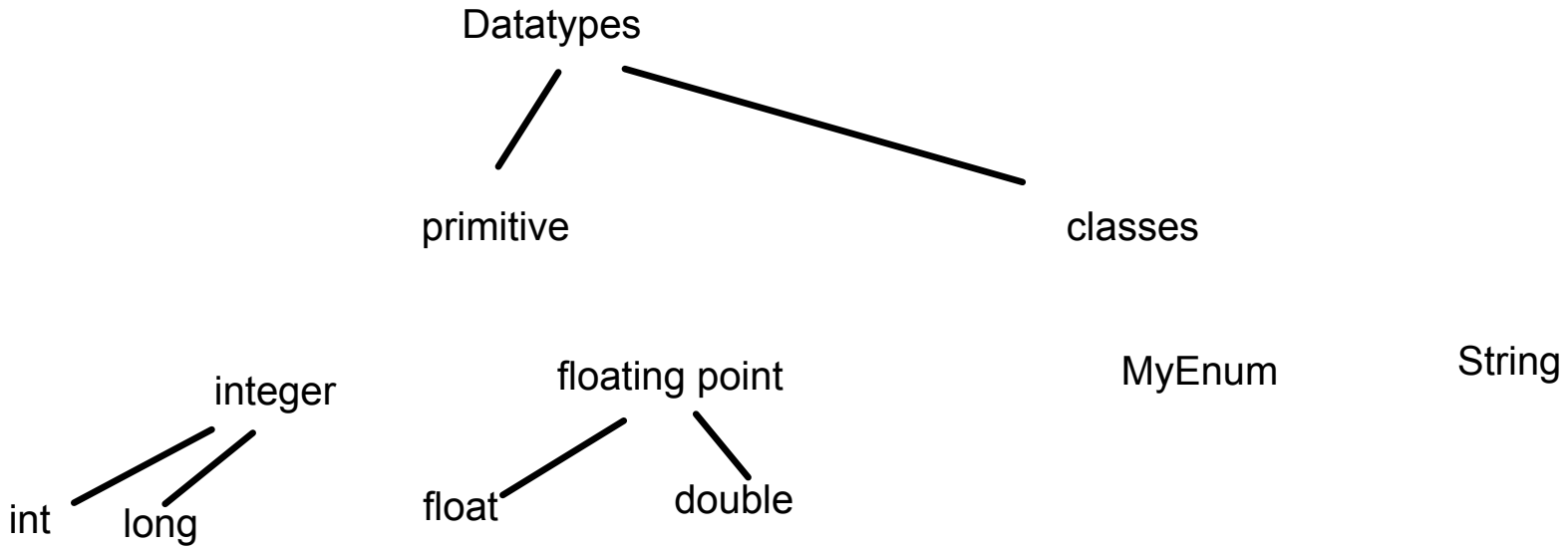
```java
package schnittstelle;

public class StartPc {
    public static void main(String[] args) {
        IGraphicsCard graphicCard = new NVidia();

        Motherboard mb = new Motherboard(graphicCard);
        mb.renderText("This is a test");
    }
}
```

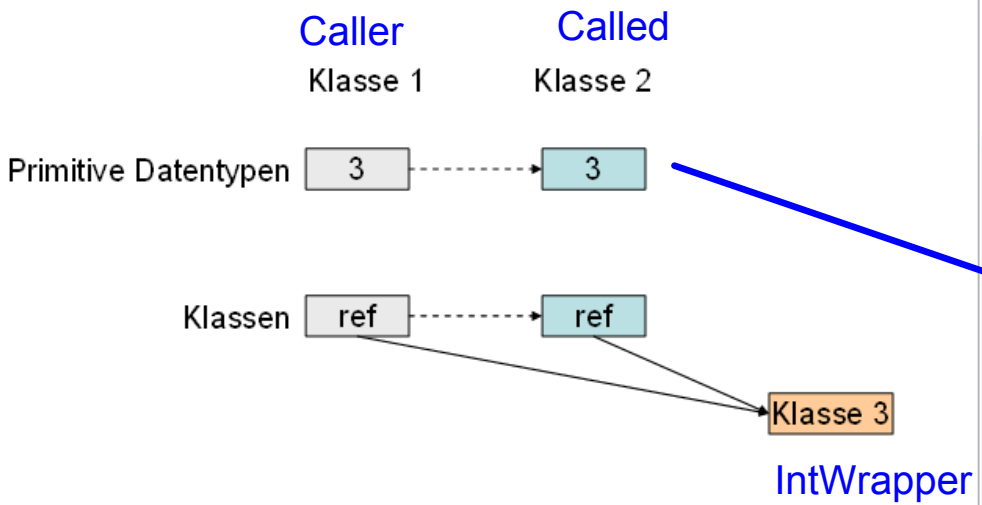
```java
package schnittstelle;

public class StartPc {
    public static void main(String[] args) {
        //IGraphicsCard graphicCard = new NVidia();

        Motherboard mb = new Motherboard(new IGraphicsCard() {

            public void display(String text) {
                System.out.println("Anonyomous inner class is displaying the text " + text);
            }

        });
        mb.renderText("This is a test");
    }
}
```

Declaration of anonymous inner class

The constructor of motherboards expects an instance of a class the implements IGraphicsCard.

We declare this class exactly at the point where it is needed. However, the class that implements the interfaces does not have a name. Therefore we call an "anonymous inner class".


Datatypes
primitive
classes
integer
floating point
MyEnum
String
int
long
float
double
call by value
call by reference



Called.java

```
3  public class Called {
4      public void increment(int i) {
5          i = i + 1;
6      }
7
8      public void increment(IntWrapper i) {
9          int oldValue = i.getB();
10         int newValue = oldValue + 1;
11         i.setB(newValue);
12     }
13
14 }
```

Motherboard.java StartPc.java AMD.java IGraphicsCard.java NV

```
3  public class Caller {
4      public static void main(String[] args) {
5          Called called = new Called();
6
7          int a = 2;
8          System.out.println("a is " + a);
9
10         called.increment(a);
11
12         System.out.println("a is " + a);
13
14         int b = 2;
15         IntWrapper wrappi = new IntWrapper(b);
16
17         called.increment(wrappi);
18
19         System.out.println("b is " + wrappi.getB());
20     }
```

IntWrapper.java

```
1  package byref;
2
3  public class IntWrapper {
4
5      private int b;
6
7      public IntWrapper(int b) {
8          this.b = b;
9      }
10
```

Implementation of equals

```java
package gleich;

public class StringWrapper {
    private String value;

    public StringWrapper(String value) {
        super();
        this.value = value;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    public boolean equals(Object o) {
        //1. Null Check
        if(o == null){
            return false;
        }

        //2. Identity check
        if(o == this) {
            return true;
        }

        //3. Type check
        if(!(o instanceof StringWrapper)){
            return false;
        }

        StringWrapper compare = (StringWrapper)o;
        if(compare.getValue().equals(this.getValue())) {
            return true;
        }
        return false;
    }
}
```

Object

typical three steps

1. null check
2. identity check
3. type check

Implementation depends on business logic

Motherboard.java | StartPc.java | AMD.java | IGraphi

```java
package gleich;

public class Car {
    private String type; //Toyota Corolla
    private String color;
    private String serialNumber;
    private String numberplate;
    private int horsePower;
    private float length;

    public boolean equals(Object o) {

        if(null == o) {
            return false;
        }

        if (o == this) {
            return true;
        }

        if(!(o instanceof Car)) {
            return false;
        }

        Car otherCar = (Car)o;
        //ferry boat logic
        if(otherCar.length == this.length){
            return true;
        }

        return false;
    }
}
```