PR2 – Formular für Lesenotizen

SS2021

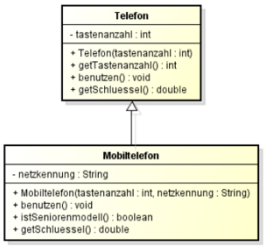
Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	30.05.21

Klasse erben

```
public class Schule {
    private String name; private String typ;
    public Schule(String name, String typ) { get und set methoden!
        this.name = name;  this.typ= typ;
    }
    public String getAbschluss() {
        if (typ.equals("Grundschule")) return "kein";
        return null;
    }
    @Override public String toString() {
        return name + " (" + typ + ")";
    }
    public void setTyp(String typ) {
        if (typ.equals("Grundschule") || typ.equals("Gymnasium")){
            this.typ= typ;
        } else {
            throw new IllegalArgumentException("Ungültiger Typ");
        }
    }
    public Schule(String name, String typ) {
        this.name = name;
        setTyp(typ);
    }
}
```

```
public class WeiterfuehrendeSchule extends Schule {
    private String zugangsvoraussetzung;
    public WeiterfuehrendeSchule(String name, String typ, String
zugangsvoraussetzung) {
        super(name, typ);
        this.zugangsvoraussetzung= zugangsvoraussetzung;
    }
    @Override public String toString() {
        return super.toString() + " [" + zugangsvoraussetzung + "]";
    }
}
```

```
@Override public double getSchluessel() {
  return 1+ super.getSchluessel();
}
```



Aufgabe Threads

```
public class AufgabeThreads {
    public static void main(String[] args) {
        final Thread c= Thread.currentThread();
        final Thread t2= new Thread() {
            @Override public void run() {
                System.out.println("Zweiter Thread");
                try {
                    c.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
        t2.start();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Thread t3= new Thread() {
            @Override public void run() {
                System.out.println("Dritter Thread");
                try {
                    t2.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Fertig"); }
        };
        t3.start();
    }
}
```

Ausgabe:
Zweiter Thread
Dritter Thread
Fertig

Aufgabe equals-Methode

```
public class Foo {
    private Boolean a;  private boolean b;

    @Override public int hashCode() {
        return java.util.Objects.hash(a,b);
    }
}
```

```
@Override public boolean equals(Object o) {
   if (o instanceof Foo ) {
       Foo temp = (Foo) o; // cast and compare it
       return temp.hashCode() == this.hashCode();
   } else {
       return false;
   }
}
```

Klasse mit Objektzähler Programmieraufgabe

```
private static int num = 0;
public static void setNum(int n) {
    num = n;
}
public Spieler(...) {
    num++;
    ...
}
```

```
public class ABCMain {
public static void main(..) {
01   A a= new A();
02   B b= new B();
03   C c= new C();
04
05   a.a();
06   b.a();     implizites upcasting
07   c.a();     implizites upcasting
08
09   a.b();     compiler error, weil a statischer & dynamischer Typ A ist und nur Methoden von A
                benutzten darf
10   b.b();
11   c.b();     implizites upcasting
12
13   a.c();     compiler error, weil a statischer & dynamischer Typ A ist und nur Methoden von A
                   benutzten darf
14   b.c();     compiler error, weil b statischer & dynamischer Typ B ist und nur Methoden von B
                   benutzten darf
15   c.c();
16
17   A x= new B();  upcast
18   x.a();
19   x.b();    compiler error, weil Compiler dynamischen Typ ignoriert & nur auf den statischen Typ
                achtet. Klasse A hat nicht die Methode b() => downcasting notwendig
20   x.c();    compiler error, weil Klasse A nicht Methode c() besitzt & downcasten bringt auch nichts
                da dynamischer Typ B ist
21
22   ((A)x).a(); typ wird nicht veraendert daher kein cast
23   ((B)x).b(); explizites downcasten
24   ((C)x).c(); runtime error, weil zu tiefes downcasten }
}
```

```
public class A {
    public void a() { System.out.println("a"); }
}
public class B extends A {
    public void b() { System.out.println("b"); }
}
public class C extends B {
    public void c() { System.out.println("c"); }
}
```

Sortieren

Comparable als Interface implementieren, soll Namen (Strings) nach Anfangsbuchstaben Sortieren
Comparator als Klasse schreiben, Namen nach Länge sortieren

```
ArrayList<Spieler> team = new ArrayList<Spieler>();

//Beispiel für Lambda-Ausdruck
Comparator<Spieler> cmp= (a, b) -> Integer.compare(a.getnum(), b.getnum());
Collections.sort(team, cmp);

Comparator<Spieler> cmpname= (a, b) -> Integer.compare(Character.toUpperCase(a.getName().charAt(0)),
                                                       Character.toUpperCase(b.getName().charAt(0)));
Collections.sort(team, cmpname);

Comparator<Spieler> namelengeth= (a, b) -> Integer.compare(a.getLength(),b.getLength());
Collections.sort(team, namelengeth);

Comparator<Spieler> cmp= new Comparator<Spieler>() {
    @Override public int compare(Spieler a, Spieler b) {
         return Integer.compare(a.getnum(), b.getnum());
    }
};


public class Spieler implements Comparable<Spieler>  {
  private String name;
  ...

  @Override
  public int compareTo(Spieler other) {
     if (Character.toUpperCase(this.getName().charAt(0)) > Character.toUpperCase(other.getName().charAt(0))) {
          return 1;
     }else if (Character.toUpperCase(this.getName().charAt(0)) < Character.toUpperCase(other.getName().charAt(0))){
         return -1;
     } else {
          return 0;
     }
  }
}
```

in main dann → `Collections.sort(team);`