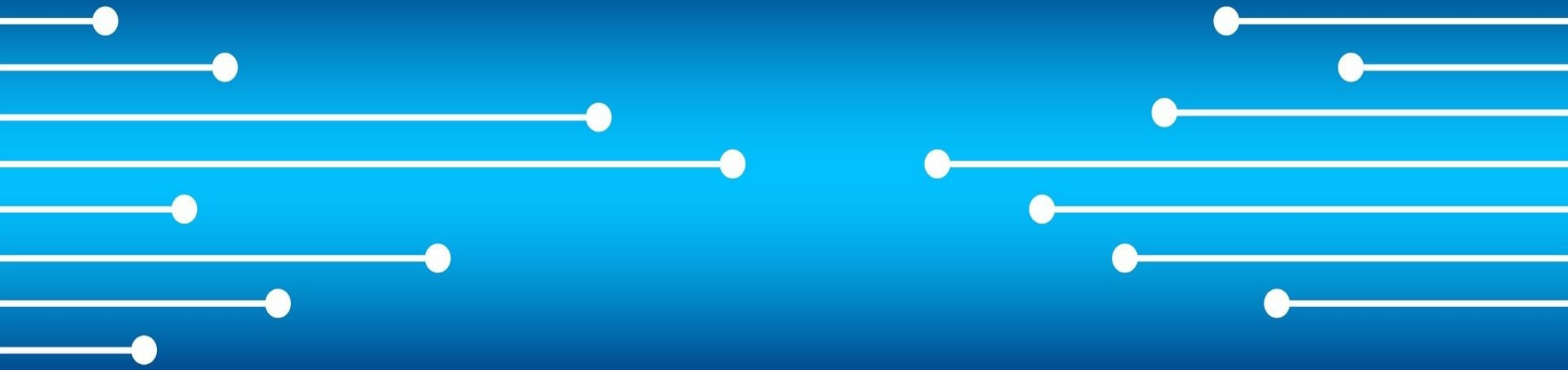
Datenverarbeitung

Teil des Moduls 5CS-DPDL-20



Prof. Dr. Deweß

Thema 8

Wichtige Grund-funktionen der Klasse „java.lang.Object“

- Überblick
- „toString“
- „equals“ und „hashCode“
- „clone“

„java.lang.Object“

- **fast alles** ist in Java ein „Object“
- ein „Object“ besitzt einige Grundfunktionen
- die von „Object“ erbenden Klassen (also alle Klassen außer „Object“) erben damit also einige Grundfunktionen, die überschrieben werden können/sollten

„java.lang.Object“ – wichtige Grundfunktionen

- **toString():**
  - erzeugt eine String-Repräsentation des Objekts
  - wird meist überschrieben, um eine inhaltliche Beschreibung des Objekts zu liefern
- **equals():**
  - für Vergleiche von Objekten
  - wird üblicherweise überschrieben, um auf inhaltliche Gleichheit zu prüfen, wobei einige Regeln eingehalten werden müssen
- **hashCode():**
  - berechnet einen Integer-Wert, der den Hash-Wert des Objekts repräsentiert und für Hashing-Verfahren und Vergleiche relevant ist
  - wird üblicherweise gemeinsam mit der equals()-Methode überschrieben
- **clone():**
  - damit „kopiert“ man Objekte

„java.lang.Object“ – wichtige Grundfunktionen

Aufgabe 1 – Vorbereitung benötigter Klassen

- Schreiben Sie zunächst rudimentär eine Klasse „Circus“. Jeder „Circus“ hat als Eigenschaften einen „name“ (vom Typ „String“) und eine „id“ (vom Primär-Typ „int“).
- Schreiben Sie im gleichen Paket zunächst rudimentär eine Klasse „SpecialCircus“, die von „Circus“ erbt („Circus“ erweitert).

  Jeder „SpecialCircus“ hat als zusätzliche Eigenschaft „hasFlag“ (vom Typ „boolean“), also noch eine Flaggge oder auch nicht.

- Schreiben Sie im gleichen Paket zunächst rudimentär eine Klasse „Clown“. Jeder „Clown“ hat als Eigenschaften einen „name“ (vom Typ „String“), einen „laughFactor“ (vom Primär-Typ „int“) und einen „circus“ (vom Typ „Circus“).

„java.lang.Object“ – wichtige Grundfunktionen

Lösung 1 – Rudimentäre Klassen

```
public class Circus {
  private String name;
  private int id;
}
```

```
public class SpecialCircus extends Circus
  private boolean hasFlag;
}
```

```
public class Clown {
  private String name;
  private int laughFactor;
  private Circus circus;
}
```

Aufgabe 2
Vervollständigen Sie diese Klassen mit:

- allen Settern und Gettern,
- einem Konstruktor, der alle Eigenschaften als Parameter entgegennimmt,
- einem Standardkonstruktor (ohne Parameter), der sinnvolle Werte setzt.

„java.lang.Object“ – wichtige Grundfunktionen

Lösung 2 – Weitere Bestandteile von „Circus“

```
public class Circus {
  //...

  public Circus() {
    this.name = "who knows";
    this.id = 0;
  }

  public Circus(String name, int id) {
    this.name = name;
    this.id = id;
  }

  //Getter und Setter
  //automatisch ...

}
```

„java.lang.Object“ – wichtige Grundfunktionen

Lösung 2 – Weitere Bestandteile von „SpecialCircus“

```
public class SpecialCircus extends Circus {
  private boolean hasFlag;

  public SpecialCircus() {
    super();
    this.hasFlag = false;
  }

  public SpecialCircus(String name, int id, boolean hasFlag) {
    super(name, id);
    this.hasFlag = hasFlag;
  }

  //Getter und Setter
  //automatisch ...

}
```

„java.lang.Object“ – wichtige Grundfunktionen

Lösung 2 – Weitere Bestandteile von „Clown“

```
public class Clown {
  //...
  public Clown() {
    this.name = "unknown";
    this.laughFactor = 0;
    this.circus = new Circus();
  }

  public Clown(String name, int laughFactor, Circus circus) {
    this.name = name;
    this.laughFactor = laughFactor;
    this.circus = circus;
  }

  //Getter und Setter automatisch ...
}
```

„java.lang.Object“ – wichtige Grundfunktionen

Aufgabe 3 – Vorbereitung einer Testklasse

Legen Sie im gleichen Paket wie die anderen Klassen eine Testklasse „ObjectMethodTest“ mit einer „main“-Methode an. Erzeugen Sie in der „main“-Methode Ihrer Testklasse folgende Objekte:

- Circus fun = new Circus("JavaFun",1);
- Circus great = new Circus("JavaFun",1);

- SpecialCircus greatWithFlag = new SpecialCircus("JavaFun",1,true);

- Clown myClown = new Clown("me",10,new Circus("JavaFun",1));

Wichtige Grund-funktionen der Klasse „java.lang.Object“

- Überblick
- **„toString“**
- „equals“ und „hashCode“
- „clone“

„toString“

- erzeugt eine String-Repräsentation des Objekts

Aufgabe und Lösung 4
Lassen Sie sich alle Objekte mit der Methode „System.out.println“ anzeigen.

```
de.baleipzig.classes.Circus@36baf30c
de.baleipzig.classes.Circus@7a81197d
de.baleipzig.classes.SpecialCircus@5ca881b5
de.baleipzig.classes.Clown@24d46ca6
```

Klassenname + @ + HashCode-Angabe

die implizit aufgerufene „toString“-Methode gehört „Object“ und über die jeweiligen Objekte als „Object“ kann man nicht viel mehr sagen;
wenn man informativere Ausgaben haben möchte, muss man die Methode in der eigenen Klasse mit informativeren, spezifischeren Angaben überschreiben

- wird meist überschrieben, um eine inhaltliche Beschreibung des Objekts zu liefern

„toString“

- erzeugt eine String-Repräsentation des Objekts

```
de.baleipzig.classes.Circus@36baf30c
de.baleipzig.classes.Circus@7a81197d
de.baleipzig.classes.SpecialCircus@5ca881b5
de.baleipzig.classes.Clown@24d46ca6
```

Klassenname + @ + HashCode-Angabe

die implizit aufgerufene „toString“-Methode gehört „Object“ und über die jeweiligen Objekte als „Object“ kann man nicht viel mehr sagen;
wenn man informativere Ausgaben haben möchte, muss man die Methode in der eigenen Klasse mit informativeren, spezifischeren Angaben überschreiben

- wird meist überschrieben, um eine inhaltliche Beschreibung des Objekts zu liefern

Aufgabe 5
Überschreiben Sie in Ihren Klassen (ggf. mit Hilfe der Entwicklungsumgebung) die Methode „toString“ für informativere Angaben und lassen Sie sich Ihre Objekte danach nochmals anzeigen.

„toString“

Lösung 5 – Beispiele für „toString“

in „Circus“

```
@Override
public String toString() {
  return "Circus [id=" + id + ", name=" + name + "]";
}
```

in „SpecialCircus“

```
@Override
public String toString() {
  return "SpecialCircus [hasFlag=" + hasFlag + ", getId()=" +
    getId() + ", getName()=" + getName() + "]";
}
```

in „Clown“

```
@Override
public String toString() {
  return "Clown [circus=" + circus + ", laughFactor=" +
  laughFactor + ", name=" + name + "]";
}
```

Wichtige Grund-funktionen der Klasse „java.lang.Object“

- Überblick
- „toString“
- **„equals“ und „hashCode“**
- „clone“

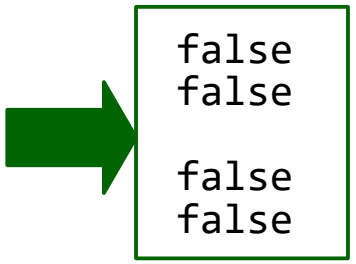
„equals“

- wird benutzt, um Objekte zu vergleichen

Aufgabe 6
Führen Sie in Ihrer Testklasse folgende Vergleiche durch:

```
System.out.println(fun.equals(great));
System.out.println(great.equals(fun));

System.out.println(greatWithFlag.equals(great));
System.out.println(great.equals(greatWithFlag));
```



Was könnte „equals“ bedeuten?

Was kann „gleich“ implementierungsunabhängig allgemein überhaupt bedeuten?

„equals“ – Was ist „gleich“?

Wann sind zwei Objekte gleich?

- wenn sie ein und dasselbe Objekt sind? ← **aktueller Referenzvergleich von „equals“**

  MEINE  ≠ DEINE  !

- wenn sie inhaltlich gleiche Objekte sind?

  Klassensatz, jeder bekommt eine Formelsammlung gestellt 

- wenn sie zwar inhaltlich nicht ganz identisch sind, aber die gleichen Eigenschaften aufweisen?

  Formelbücher, keine Äpfel 

- wenn sie jetzt gleich sind oder wenn sie für immer gleich sind?
- bei komplexeren Objekten (z.B. „Collections“):
  Spielt die Reihenfolge der enthaltenen Objekte eine Rolle?

„equals“ – Mindestanforderungen?

„gleich“ mag nicht „gleich“ sein, aber:

Was sollten „gleich“ und „gleich“ gemeinsam haben?

- ein Objekt sollte sich selbst gleich sein:
  clownX.equals(clownX) sollte „true“ ergeben

reflexiv

- Gleichheit sollte nicht abhängig von der Vergleichsrichtung sein:
  wenn clownX.equals(clownY) „true“ ist,
  dann sollte clownY.equals(clownX) auch „true“ sein

symmetrisch

- „gleich“ und „gleich“ ist „gleich“:
  wenn clownX.equals(clownY) „true“ ist und
  clownY.equals(clownZ) „true“ ist,
  dann sollte auch clownX.equals(clownZ) „true“ sein

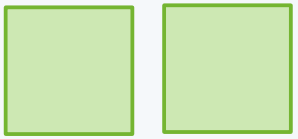
transitiv

- „gleich“ sollte also reflexiv, symmetrisch und transitiv sein

eine Äquivalenzrelation

„equals“

„quadratisches“ Rechteck
gleich oder ungleich Quadrat?



Wann sind zwei Objekte in Java gleich?

Aufgabe 7
Überschreiben Sie in Ihren Klassen (ggf. mit Hilfe der Entwicklungsumgebung) die Methoden „equals“ und „hashCode“ (schauen wir uns später an, brauchen wir aber hier schon) und schauen Sie sich nun die Vergleiche aus Aufgabe 6 wieder an.

```
System.out.println(fun.equals(great));
System.out.println(great.equals(fun));

System.out.println(greatWithFlag.equals(great));
System.out.println(great.equals(greatWithFlag));
```

```
true
true

false
false
```

- was in Java gleich ist, hängt von der Implementierung von „equals“ ab
- bei der bei den meisten Entwicklungsumgebungen gewählten Implementierung (Sie könnten anders!) können Objekte einer Klasse nie Objekten einer Subklasse gleichen
- dies widerspricht eigentlich dem Prinzip, dass Subklasse-Objekte auch Objekte der Superklasse sind → besonderes Augenmerk auf „gleich“ bzw. auf Vererbungsstrukturen legen, da sich dieser Konflikt zwischen „was ist gleich“ und „was wollen wir für Super-Sub-Typ-Beziehungen“ nicht einfach (oder gar nicht?) auflösen lässt!

„hashCode“

- berechnet einen Integer-Wert, der den Hash-Wert des Objekts repräsentiert und für Hashing-Verfahren und Vergleiche relevant ist
- sollte konsistent mit sich selbst sein, solange sich das Objekt nicht ändert:

  clownX.hashCode() == clownX.hashCode
- muss zu „equals“ passen (und deshalb bei „equals“ auch überschrieben werden):
  - wenn clownX.equals(clownY) „true“ ist, muss clownX.hashCode() gleich clownY.hashCode() sein
  - wenn clownX.hashCode() gleich clownY.hashCode() ist, muss aber nicht clownX.equals(clownY) „true“ sein;

    es ist eher so, dass der hashCode() als performancestarker Vorfilter vor dem eigentlichen Vergleich genutzt wird; wenn die hashCodes gleich sind, dann könnten die Objekte gleich sein, sonst nicht

    (so wie zwei Bilddateien Kopien voneinander sein könnten, wenn Sie genauso groß sind, und sonst nicht)

„hashCode“ – am Beispiel

automatisch generierte „hashCode()“

```
public class Circus {
  //...

  @Override
  public int hashCode() {
    return Objects.hash(id, name);
  }
}
```

Was könnte noch die Anforderungen an „hashCode()“ erfüllen?

```
public int hashCode() {
  return 42;
}
```

gleiche „Circus“-Objekte haben damit einen gleichen hashCode, aber da alle den gleichen haben, wäre das ein schlechter Vor-Filter für einen Gleichheitstest

```
public int hashCode() {
  return id;
}
```

gleiche „Circus“-Objekte haben damit einen gleichen hashCode, aber da der „name“ noch nicht berücksichtigt wird, wäre das (insbesondere im Vergleich zur automatisch generierten Methode) auch kein guter Vor-Filter für einen Gleichheitstest

Wichtige Grund-funktionen der Klasse „java.lang.Object“

- Überblick
- „toString“
- „equals“ und „hashCode“
- **„clone“**

„clone“

- damit kopiert man prinzipiell Objekte
- ist standardmäßig geschützt (protected) und setzt außerdem voraus, dass die Klasse des aufrufenden Objekts die methodenlose Schnittstelle „Cloneable“ interpretiert
- zum öffentlichen „Clonen“ (Erlaubnis zum „Clonen“ über „protected“ [nur Subklassen und Klassen im gleichen Paket dürften „clonen“] hinaus) muss daher „clone()“ in der eigenen Klasse mit Zugriffsmodifikator „public“ überschrieben werden und die eigene Klasse muss formal im Sinne eines Markers „Cloneable“ implementieren

Aufgabe 8
Lassen Sie sich mit Hilfe der Entwicklungsumgebung in Ihren Klassen „Clown“ und „Circus“ die Methode „clone“ überschreiben, erweitern Sie Ihren Klassenkopf (sofern nicht automatisch geschehen) um die Angabe `implements Cloneable` damit klar ist, dass Ihre Klasse nun „clonebare“ Objekte repräsentiert, und passen Sie ggf. den Zugriffsmodifizierer auf `public` an.

Hinweis:
bei Eclipse unter „Quelltext“ unter „Methoden überschreiben/implementieren“ nachschauen

„clone“

Ein erster „Clown-Clone“

Aufgabe 9
Ergänzen Sie Ihre Testklasse zunächst um:

```
Clown anotherClown = null;
try {
  anotherClown =  (Clown) myClown.clone();
} catch (CloneNotSupportedException e) {
  e.printStackTrace();
}
```

Lassen Sie sich nun die beiden Clowns in Ihrer Klasse auf der Konsole anzeigen.

„clone“

Ein erster „Clown-Clone“

Lösung 9

```
System.out.println("my Clown: " + myClown);
System.out.println("another Clown: " + anotherClown);
```

```
my Clown: Clown [circus=Circus [id=1, name=JavaFun], laughFactor=10, name=me]
another Clown: Clown [circus=Circus [id=1, name=JavaFun], laughFactor=10, name=me]
```

Sieht gut aus – freuen Sie sich aber nicht zu früh ;-)

„clone“ – „flache“oder „tiefe“ Kopie?

Aufgabe 10 - Was es wirklich mit unseren „Clowns“ auf sich hat …

Unser geclonter „Clown“ verlässt sein Spiegelbild und will eigene Wege gehen. Ändern Sie in Ihrer Testklasse nun Ihren geclonten „Clown“ sinngemäß wie folgt ab

```
anotherClown.setName("you");
anotherClown.setLaughFactor(5);
anotherClown.getCircus().setId(2);
```

und lassen Sie sich nun die beiden „Clowns“ auf der Konsole anzeigen.

„clone“ – „flache“ Kopie

Unser erster „Clown-Clone“

Lösung 10

```
System.out.println("my Clown: " + myClown);
System.out.println("another Clown: " + anotherClown);
```

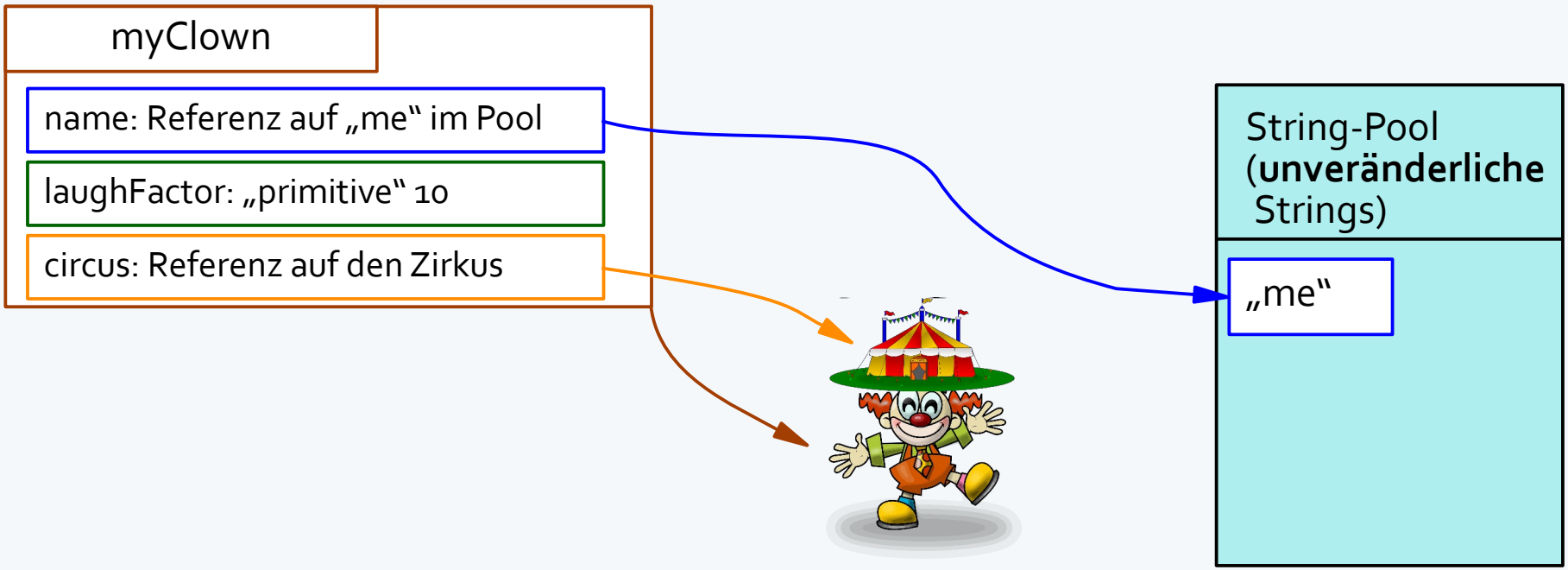
```
my Clown: Clown [circus=Circus [id=2, name=JavaFun], laughFactor=10, name=me]
another Clown: Clown [circus=Circus [id=2, name=JavaFun], laughFactor=5, name=you]
```

So wollten wir das nicht, oder?

„Flache“ Kopie:
Inhalte der Variablen wurden einfach kopiert, aber nicht ggf. sich dahinter verbergende Objekte.

„clone“ – „flache“ Kopie

Unser erster „Clown-Clone“



„clone“ – „flache“ Kopie

Unser erster „Clown-Clone“

myClown

name: Referenz auf „me“ im Pool

laughFactor: „primitive“ 10

circus: Referenz auf den Zirkus

„clonen“

anotherClown

name: Referenz auf „me“ im Pool

laughFactor: „primitive“ 10

circus: Referenz auf den Zirkus

String-Pool (**unveränderliche** Strings)

„me“

„clone“ – „tiefe“ Kopie

Ein zweiter Versuch

Aufgabe 11
Wir müssen offensichtlich auch den „Circus“ clonen.
Ändern Sie entsprechend den Quellcode von „clone“ in Ihrer „Clown“-Klasse wie folgt ab:

```
@Override
public Object clone() throws CloneNotSupportedException {
  //return super.clone();
  Clown clonedClown = (Clown) super.clone();
  clonedClown.circus = (Circus) circus.clone();
  return clonedClown;
}
```

Lassen Sie sich nun die beiden Clowns in Ihrer Klasse auf der Konsole anzeigen.

„clone“ – „tiefe“ Kopie

Ein zweiter „Clown-Clone“

Lösung 11

```
System.out.println("my Clown: " + myClown);
System.out.println("another Clown: " + anotherClown);
```

```
my Clown: Clown [circus=Circus [id=1, name=JavaFun], laughFactor=10, name=me]
another Clown: Clown [circus=Circus [id=2, name=JavaFun], laughFactor=5, name=you]
```

Jetzt können Sie sich freuen ;-)

„Tiefe“ Kopie:
Inhalte der Variablen und sich ggf. dahinter verbergende Objekte wurden kopiert.
(wobei wir eigentlich noch nicht „tief“ kopiert haben [String-Objekt-Kopie fehlt], was aber praktisch egal ist, weil Strings in Java unveränderlich sind)

„clone“ – Bemerkungen

- bei einem einfachen „Clone“-Vorgang wird standardmäßig „flach“ kopiert, es wird nur ein neues Objekt erzeugt und die Variableninhalte werden kopiert, ggf. sich hinter Referenzen verbergende Objekte werden aber nicht kopiert, so dass die Kopien nicht unabhängig voneinander sind

- bei einer „tiefen“ Kopie werden auch sich ggf. hinter Referenzen verbergende Objekte kopiert, was allerdings bei aufwendigen Vererbungsstrukturen mit „clone“ sehr aufwendig sein kann, weil dann in jeder Klasse entsprechend „tiefe“ Implementierungen von „clone“ geschrieben werden müssen, was ggf. (fremder Code?) nicht möglich ist

„clone“ – Alternativen?

Wenn man eine aufwendige Vererbungsstruktur hat, mit viel Fremdcode arbeitet oder fehlertolerant Erweiterungen erwartet usw. kann man statt „clone“ auch …

- ggf. Konstruktoren speziell zum Kopieren anlegen, bei dem Objekte basierend auf den vorhandenen Objekten neu erzeugt werden, z.B.:

```
public Clown(Clown clownToCopy) {
  this(clownToCopy.getName(), clownToCopy.getLaughFactor(),
        new Circus(clownToCopy.getCircus()));
}
```

sowie

```
public Circus(Circus circusToCopy) {
  this(circusToCopy.getName(), circusToCopy.getId());
}
```

- sofern möglich, ein Objekt serialisieren und in ein neues Objekt deserialisieren, z.B. mittels der externen „GSON“-Bibliothek zur JSON-Serialisierung

Vielen Dank für Ihre Aufmerksamkeit.