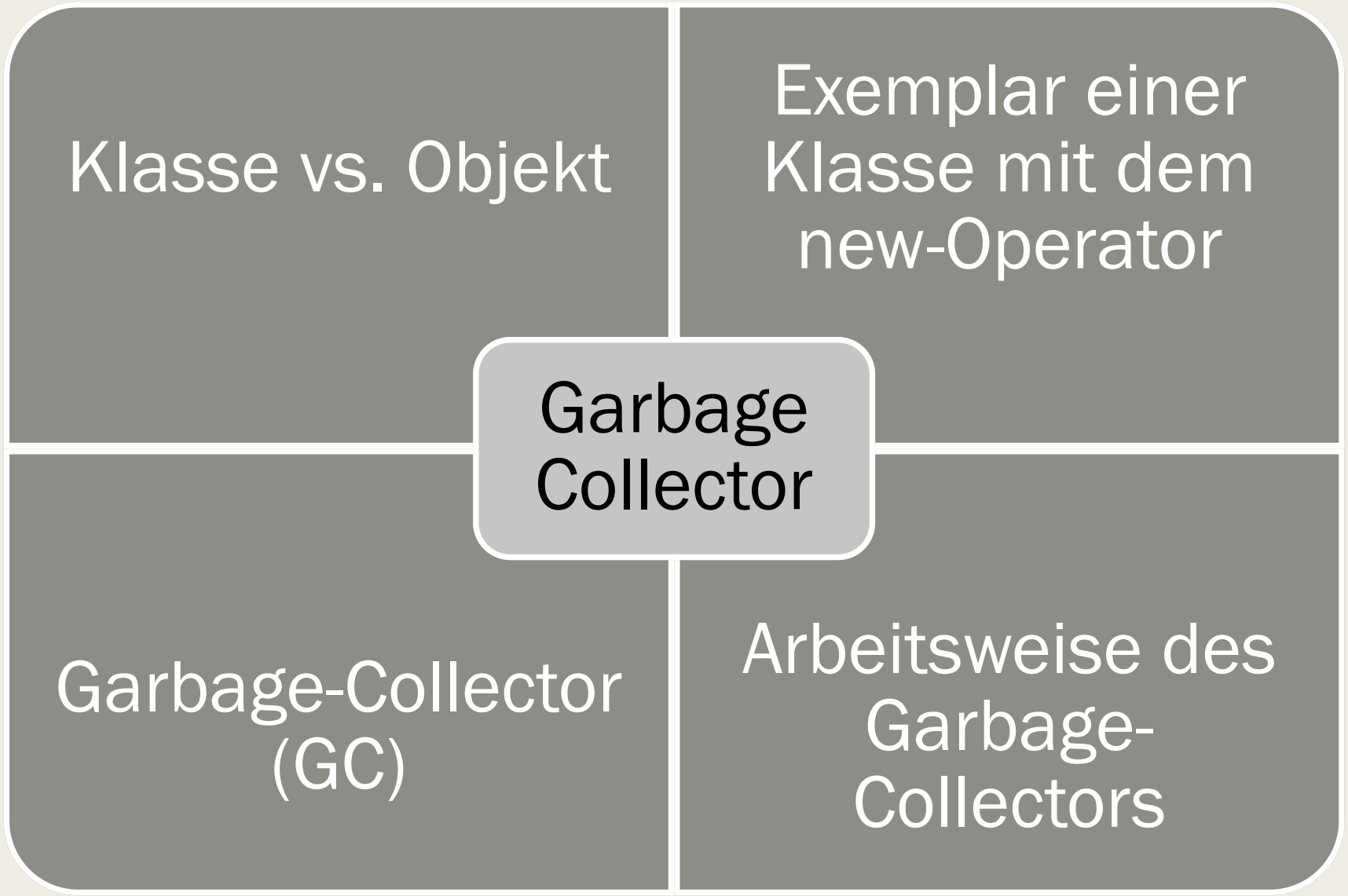
GARBAGE COLLECTOR

Christian Schirmer


Klasse vs. Objekt
Exemplar einer Klasse mit dem new-Operator
Garbage Collector
Garbage-Collector (GC)
Arbeitsweise des Garbage-Collectors

Klasse vs. Objekt

- Klasse: beschreibt Aussehen eines Objekts
- Objekte entsprechen Elementen und Klassen den Mengen, in denen die Objekte enthalten sind
- Diese Objekte haben Eigenschaften
- Es gibt Möglichkeiten, diese Zustände zu erfragen und zu ändern

Christian Schirmer
Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- neue Objekte in Java: ausdrücklich mit new-Operator

```
new java.awt.Point();
```

- new-Operator wird gefolgt von Name der Klasse
- Klassenname (hier voll qualifiziert) - Point befindet sich im Paket java.awt
  - *Paket: Gruppe zusammengehöriger Klassen Schreibweise kann durch imports gekürzt werden Hinter dem Klassennamen sind runde Klammern für den Konstruktor Aufruf*

Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- **Konstruktoraufruf**: eine Art Methodenaufruf
  - *Es werdenWerte für die Initialisierung des frischen Objekts übergeben*
- Speicherverwaltung von Java reserviert für neues Objekt freien Speicher
- Gültiger Aufruf des Konstruktors:
  - *new-Ausdruck gibt Referenz auf Objekt zurück*

Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- **new** Operator:
  - *Laufzeitsystem reserviert es so viel Speicher, dass alle Objekteigenschaften und Verwaltungsinformationen Platz finden*
  - *Speicherplatz nimmt Laufzeitumgebung vom Heap.*
  - *(vordefinierte Maximalgröße: nicht beliebig viel Speicher vom Betriebssystem abgreifbar)*

Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- Alternativen zum new Operator:
  - *newInstance()*
  - *clone() neues Objekt als Kopie eines anderen Objekts erzeugen*
  - *String-Konkatenation mit + : Compiler setzt **new** ein um neues String-Objekt anzulegen*

Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- System nicht in der Lage, genügend Speicher für ein neues Objekt bereitzustellen:
  - *Garbage-Collector versucht in letzter Rettungsaktion, alles Ungebrauchte wegzuräumen*
  - *Immer noch nicht ausreichend Speicher frei: Laufzeitumgebung generiert OutOfMemoryError und bricht Abarbeitung ab*

Christian Schirmer

Exemplar einer Klasse mit dem new-Operator

- Heap- Speicher
  - *Hier werden Objekt Variablen und Instanzen gespeichert*
- Stack – Speicher (Stapelspeicher)
  - *Lokale Variablen*
  - *Methodenaufruf mit Variablen: Argumente kommen vorm Methodenaufruf auf den Stapel*
  - *Methode kann über Stack auf die Werte lesend oder schreibend zugreifen.*
  - *Endlose rekursive Methodenaufrufe: ist maximale Stack-Größe erreicht kommt es zu Exception vom Typ java.lang.StackOverflowError*
  - *Mit jedem Thread ein JVM-Stack assoziiert: bedeutet das das Ende des Threads auch das ende des Stacks darstellt*

Christian Schirmer

GARBAGE-COLLECTOR (GC)

Christian Schirmer

Garbage-Collector (GC)

- GC ist ein nebenläufiger Thread
- GC testet ob Objekte auf dem Heap noch benötigt werden
- Objekte nicht benötigt -> sie werden gelöscht
- Objekt nicht referenziert (benötigt): Garbage- Collector (GC) gibt reservierten Speicher frei
- java-Schalter –verbose:gc
  Konsolenausgaben, wenn GC nicht referenzierte Objekte erkennt und wegräumt
- Wegnahme der letzten Referenz: Tod des Objekts

Christian Schirmer

Garbage-Collector (GC)

- Einsatz eines GC verhindert zwei Probleme:
  1. *Ein Objekt kann gelöscht werden, aber die Referenz existiert noch (engl. dangling pointer)*
  2. *Kein Zeiger verweist auf ein bestimmtes Objekt, dieses existiert aber noch im Speicher (engl. memory leak)*
- Dekonstruktoren kennt Java nicht
- finalize()-Methode: Ähnlich zum Dekonstruktor

Christian Schirmer

ARBEITSWEISE DES GARBAGE-COLLECTORS (GC)

Christian Schirmer

Arbeitsweise des Garbage-Collectors (GC)

- GC ist ein unabhängiger Thread mit niedriger Priorität
- GC verwaltet Wurzelobjekte, von denen aus das gesamte Geflecht der lebendigen Objekte (der sogenannte Objektgraph) erreicht werden kann
  - *Wurzel des Thread-Gruppen-Baums*
  - *lokalen Variablen aller aktiven Methodenaufrufe (Stack aller Threads).*
- GC markiert und entfernt nicht benötigte Objekte
- HotSpot-Technologie (Objekte Anlegen - sehr schnell)
- HotSpot verwendet generationenorientierten GC
  - *Zwei Gruppen von Objekten mit unterschiedlicher Lebensdauer (meisten Objekte sterben sehr jung, die wenigen werden sehr alt)*

Christian Schirmer

Arbeitsweise des Garbage-Collectors (GC)

- Strategie:
  - *Objekte werden im »Schnelllebigen Bereich« erzeugt, dieser wird oft nach toten Objekten durchsucht und ist in der Größe beschränkt*
  - *Überlebende Objekte kommen nach einiger Zeit aus dem Kindergarten in eine andere Generation »Langlebigen Bereich«*
  - *Diese wird vom GC selten durchsucht*
- GC arbeitet ununterbrochen und räumt auf.
- Beginnt nicht erst mit der Arbeit, wenn es zu spät und der Speicher schon voll ist

Christian Schirmer
Christian Schirmer

Arbeitsweise des Garbage-Collectors (GC)

- Szenario: GC wird nicht mehr benötigtes Objekt hinter der Referenzvariablen ref entfernen, wenn die Laufzeitumgebung den inneren Block verlässt

```
{
    Kuh ref = new Kuh();
}
```

- Nach dem Block ist ref für die Garbage Collection frei.

Christian Schirmer

Arbeitsweise des Garbage-Collectors (GC)

- In fremden Programmen sind mitunter Anweisungen wie die folgende zu lesen:

```
ref = null;
```

- Oft unnötig: GC weiß, wann der letzte Verweis vom Objekt genommen wurde.
- Anders, wenn Lebensdauer der Variablen größer (Objekt- oder statischen Variablen) oder wenn diese in einem Feld referenziert

Christian Schirmer

Arbeitsweise des Garbage-Collectors (GC)

- Wird referenziertes Objekt nicht mehr benötigt:
  - *Variable (oder der Feldeintrag) sollte mit null belegt werden*
  - *GC würde andernfalls das Objekt aufgrund der starken Referenzierung nicht wegräumen*
  - *GC findet jedes nicht referenziertes Objekt*
  - *Fähigkeit zur Divination (Wahrsagen), Speicherlecks durch unbenutzte, aber referenzierte Objekte aufzuspüren hat GC nicht*

Christian Schirmer

REFERNZ MIT NULL BELEGEN

Christian Schirmer

Referenz mit null belegen

```java
public class Bauernhofsimulator {

    public static void main(String[] args) {
        Kuh kuh = new Kuh();
        System.out.println(kuh);
        // kuh ist immer noch nicht frei für die GC

        kuh = null;
        // jetzt ist kuh für die GC frei
        // the object is eliggble for Garbage Collection
    }

}
```


Christian Schirmer

LEBENSZEIT VON OBJEKTEN (METHODEN)

Christian Schirmer

```
import java.util.Date;
public class GarbageFactory {
   public static void main(String [] args) {
      Date d = getDate();
      doComplicatedStuff();
      System.out.println("d = " + d);
   }

   public static Date getDate() {
      Date d2 = new Date();
      StringBuffer now = new StringBuffer(d2.toString());
      System.out.println(now);
      return d2;
   }
```

Christian Schirmer

GC EXPLIZIT AUFRUFEN

Christian Schirmer

GC explizit aufrufen

```
import java.util.Date;

public class CheckGC {

    public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        System.out.println("Total JVM memory : " + rt.totalMemory());

        System.out.println("Before memory\t : " + rt.freeMemory());

        Date d = null;
        for (int i = 0; i < 10_000_000; i++) {
            d = new Date();
            d = null;
        }


        System.out.println("After memory\t : " + rt.freeMemory());

        //System.gc(); // Aufruf der Garbage Collection
        rt.gc(); // Alternativ Aufruf der Garbage Collection

        System.out.println("After GC memory\t : " + rt.freeMemory());
    }

}
```

```
Total JVM memory  : 126877696
Before memory     : 126206592
After memory      : 119271912
After GC memory   : 125924864
```


Christian Schirmer

METHODE FINALIZE

Christian Schirmer

Methode finalize()

- Methode finalize() ist in der Klasse Object definiert
- Folglich besitzt jedes Objekt eine geerbte Methode finalize()
- Methode finalize() der Klasse Object hat allerdings einen leeren Methodenrumpf. Der Garbage Collector ruft die Methode finalize() auf, bevor er ein Objekt aus dem Speicher entfernt.
- Durch Überschreiben der Methode finalize() könnten beispielsweise nicht mehr benötigte Ressourcen für ein Objekt freigegeben werden.
- Da weder definiert, wann der Garbage Collector Objekte aus dem Speicher entfernt, noch sichergestellt ist, dass der Garbage Collector auf jeden Fall eine Speicherbereinigung vor der Beendigung eines Programms durchführt, ist generell von der Verwendung der Methode finalize() abzuraten.

Christian Schirmer

Methode finalize()

- ACHTUNG....
  - *finalize() wird nur ein einziges mal je Objekt aufgerufen.*
  - *Indem innerhalb der Methode eine Referenz auf dieses Objekt gesetzt und nach draußen gereicht wird, wäre das Objekt nicht mehr Frei für die GC.*
  - *Die GC merkt sich, das finalize() für dieses Objekt schon aufgerufen wurde und wird finalize() für dieses Objekt nicht mehr ausführen*

Christian Schirmer