Neue Themen für die 808

Februar 2025

SMART INDUSTRY CAMPUS

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Selbsteinschätzung • Heap und Stack • Garbage Collector und Object Lifecycle	• Exceptions	• StringBuilder vs. String	• Calender API	• Wiederholung ArrayLists • Lambdas

SMART INDUSTRY CAMPUS

Plan für heute

- Selbsteinschätzung ausfüllen lassen
- Stack und Heap
- GarbageCollection
- Object Lifecycle

SMART INDUSTRY CAMPUS

Stacks und Heaps

SMART INDUSTRY CAMPUS

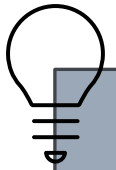
Grundlagen

- Java verwaltet Speicher automatisiert, indem es Stack und Heap nutzt
- Wo werden Methoden und Variablen gespeichert?

SMART INDUSTRY CAMPUS

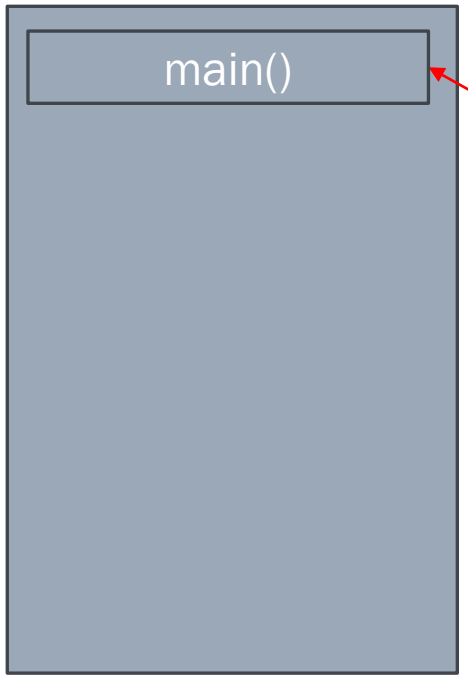
Stack-Speicher

- wird für Methodenaufrufe und lokale Variablen genutzt
- organisiert nach LIFO-Prinzip (LastIn, FirstOut)
- Speicherfreigabe beim Verlassen einer Methode erfolgt automatisch
- kann jedoch auch überfüllt werden -> StackOverflowException



LIFO:
Das letzte Element, das hinzugefügt wird, wird als erstes entfernt

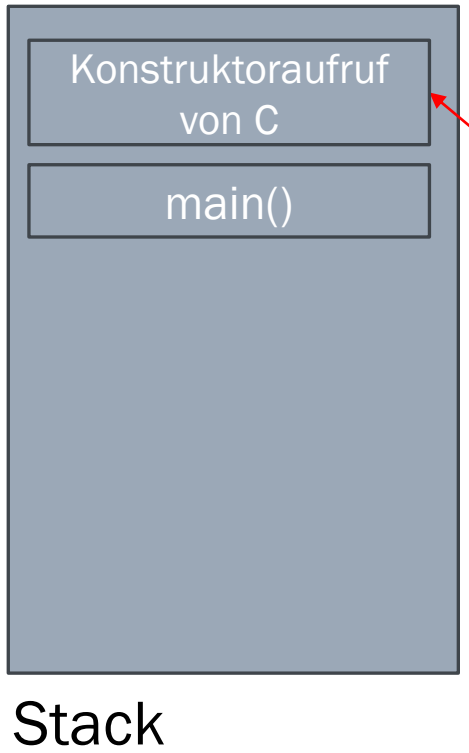
Stack-Speicher – Beispi



Stack

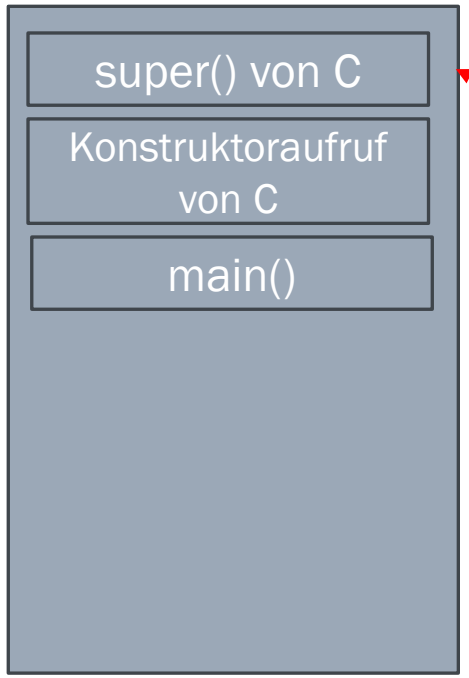
```
class A{
   A(){
      System.out.println("Hallo ich bin Klasse A");
   }
}

class B extends A{
   B(){
      System.out.println("Hallo ich bin Klasse B");
   }
}

class C extends B{
   C(){
      System.out.println("Hallo ich bin Klasse C");
   }
}
public class StackExample {
   public static void main(String[] args) {
      C c;                  //Referenz c im Stack
      c = new C();      //Objekt wird erzeugt (Heap)
   }
}
```

Stack-Speicher – Beispi




```java
class A{
   A(){
      System.out.println("Hallo ich bin Klasse A");
   }
}

class B extends A{
   B(){
      System.out.println("Hallo ich bin Klasse B");
   }
}

class C extends B{
   C(){
      System.out.println("Hallo ich bin Klasse C");
   }
}
public class StackExample {
   public static void main(String[] args) {
      C c;                    //Referenz c im Stack
      c = new C();            //Objekt wird erzeugt (Heap)
   }
}
```

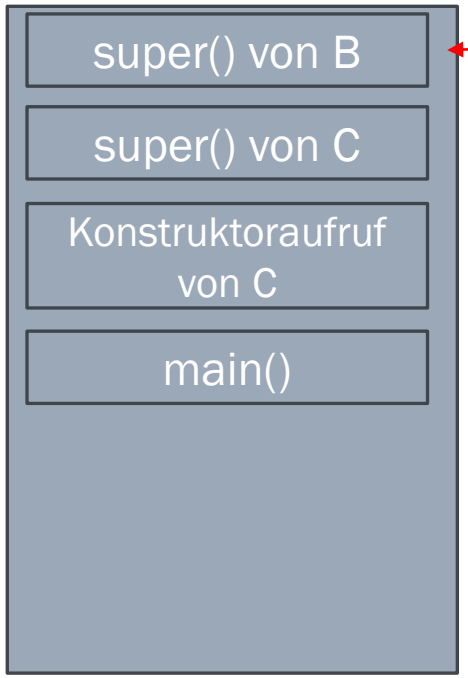
Stack-Speicher – Beispi



Stack

```
class A{
   A(){
      System.out.println("Hallo ich bin Klasse A");
   }
}

class B extends A{
   B(){
      System.out.println("Hallo ich bin Klasse B");
   }
}

class C extends B{
   C(){
      System.out.println("Hallo ich bin Klasse C");
   }
}
public class StackExample {
   public static void main(String[] args) {
      C c = new C();
   }
}
```

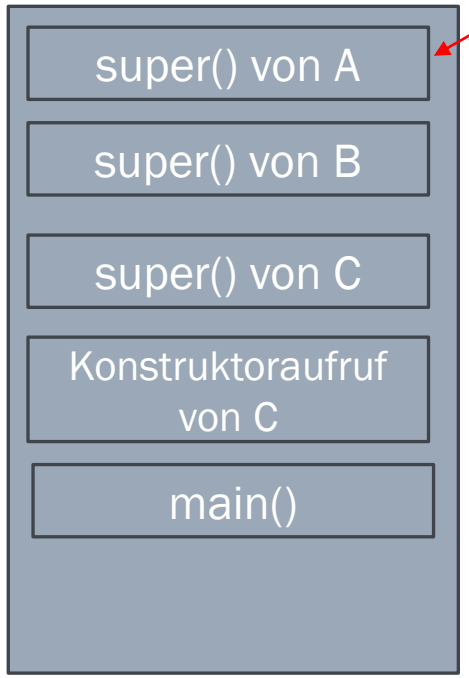
Stack-Speicher – Beispi



Stack

```
class A{
    A(){
        System.out.println("Hallo ich bin Klasse A");
    }
}

class B extends A{
    B(){
        System.out.println("Hallo ich bin Klasse B");
    }
}

class C extends B{
    C(){
        System.out.println("Hallo ich bin Klasse C");
    }
}
public class StackExample {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

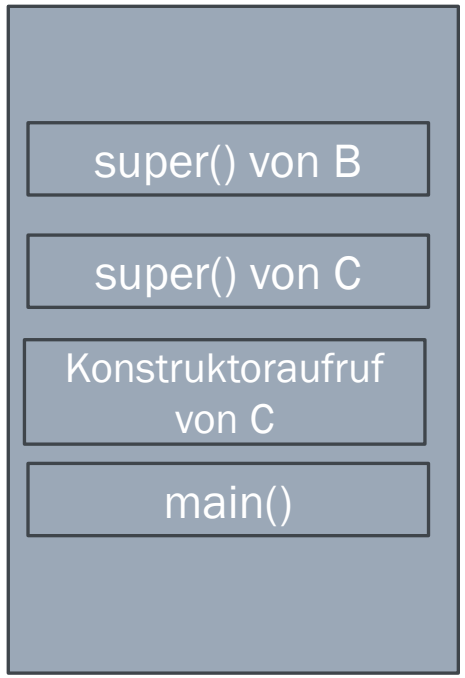
Stack-Speicher – Beispi



Stack

```java
class A{
    A(){
        System.out.println("Hallo ich bin Klasse A");
    }
}

class B extends A{
    B(){
        System.out.println("Hallo ich bin Klasse B");
    }
}

class C extends B{
    C(){
        System.out.println("Hallo ich bin Klasse C");
    }
}
public class StackExample {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

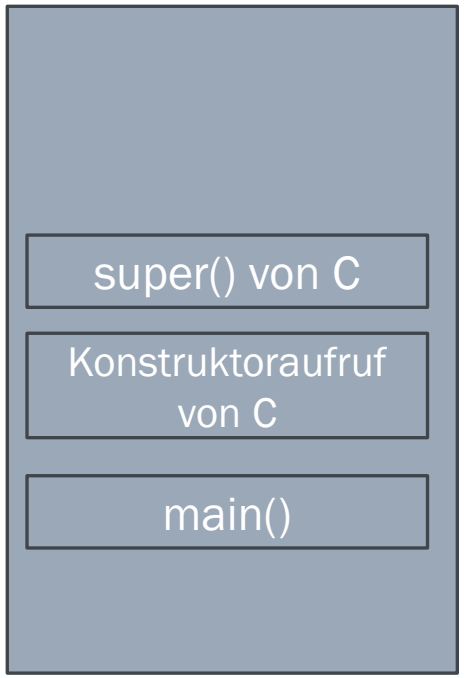
Stack-Speicher – Beispi



Stack

```
class A{
   A(){
      System.out.println("Hallo ich bin Klasse A"); //Ausgabe
   }
}

class B extends A{
   B(){
      System.out.println("Hallo ich bin Klasse B");
   }
}

class C extends B{
   C(){
      System.out.println("Hallo ich bin Klasse C");
   }
}
public class StackExample {
   public static void main(String[] args) {
      C c = new C();
   }
}
```

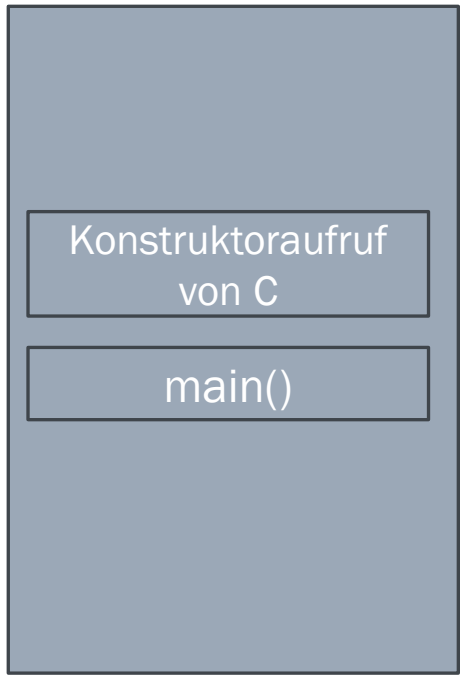
Stack-Speicher – Beispi



Stack

```
class A{
   A(){
      System.out.println("Hallo ich bin Klasse A");
   }
}

class B extends A{
   B(){
      System.out.println("Hallo ich bin Klasse B"); //Ausgabe
   }
}

class C extends B{
   C(){
      System.out.println("Hallo ich bin Klasse C");
   }
}
public class StackExample {
   public static void main(String[] args) {
      C c = new C();
   }
}
```

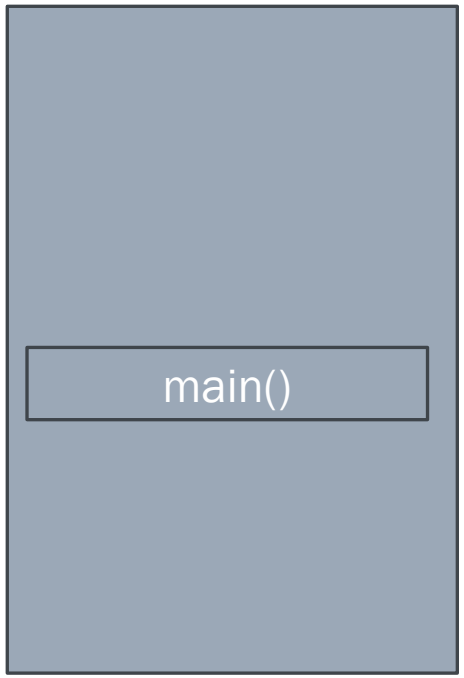
Stack-Speicher – Beispi



Stack

```
class A{
    A(){
        System.out.println("Hallo ich bin Klasse A");
    }
}

class B extends A{
    B(){
        System.out.println("Hallo ich bin Klasse B");
    }
}

class C extends B{
    C(){
        System.out.println("Hallo ich bin Klasse C"); //Ausgabe
    }
}
public class StackExample {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

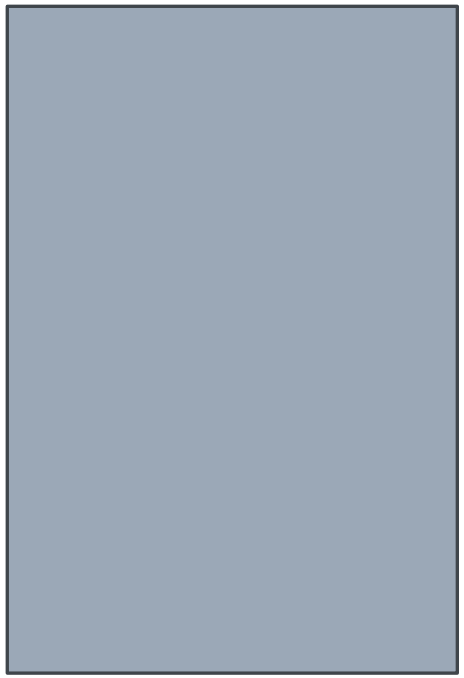
Stack-Speicher – Beispi



Stack

```
class A{
    A(){
        System.out.println("Hallo ich bin Klasse A");
    }
}

class B extends A{
    B(){
        System.out.println("Hallo ich bin Klasse B");
    }
}

class C extends B{
    C(){
        System.out.println("Hallo ich bin Klasse C"); //Ausgabe
    }
}
public class StackExample {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

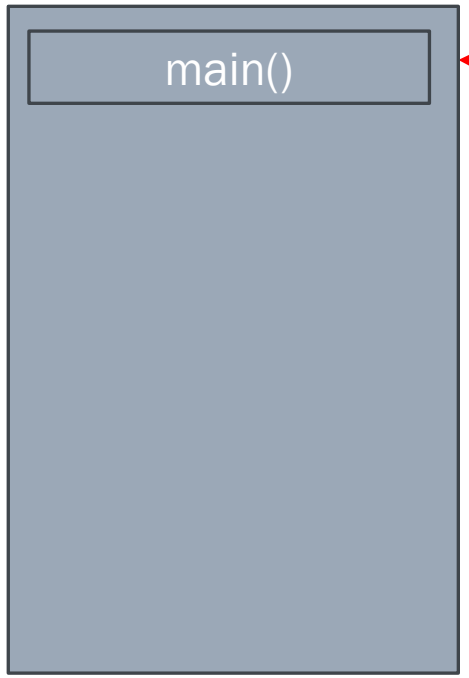
Stack-Speicher – Beispi



Stack

```
class A{
    A(){
        System.out.println("Hallo ich bin Klasse A");
    }
}

class B extends A{
    B(){
        System.out.println("Hallo ich bin Klasse B");
    }
}

class C extends B{
    C(){
        System.out.println("Hallo ich bin Klasse C"); //Ausgabe
    }
}
public class StackExample {
    public static void main(String[] args) {
        C c = new C();
    }
}
```

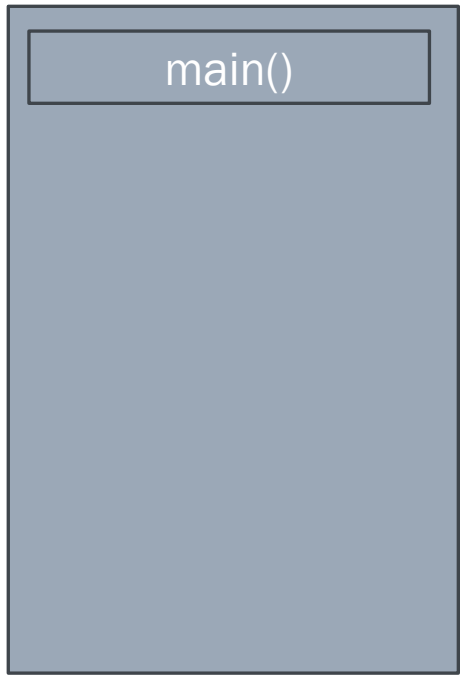
Stack-Speicher - Beispiel



Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

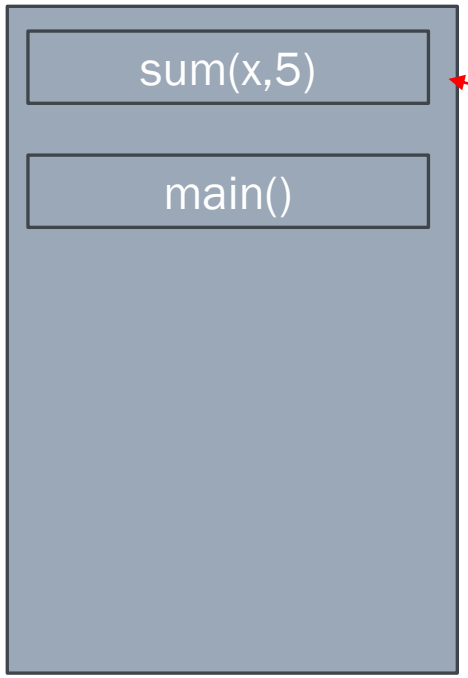
Stack-Speicher - Beispiel



Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;                    //wird deklariert und initialisiert
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

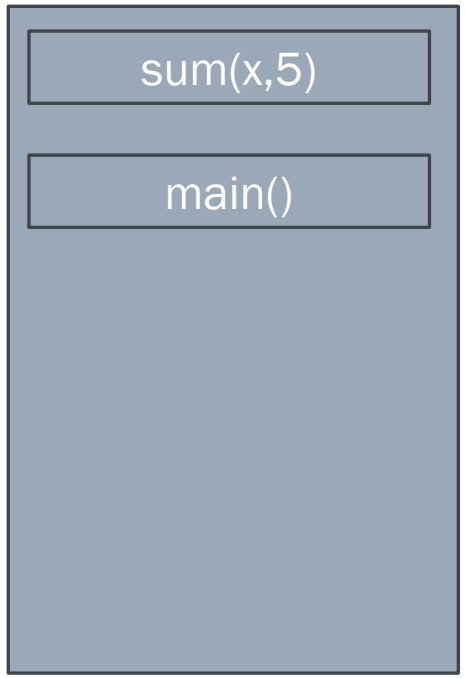
Stack-Speicher - Beispiel



Stack

```java
public class StackExample {
    public static void main(String[] args) {
        int x = 10;                    //wird deklariert und initialisiert
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

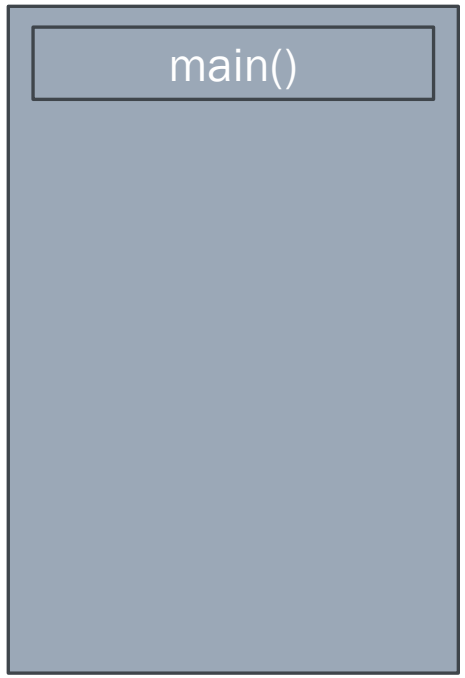
Stack-Speicher - Beispiel



Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;                    //Rückgabe
    }
}
```

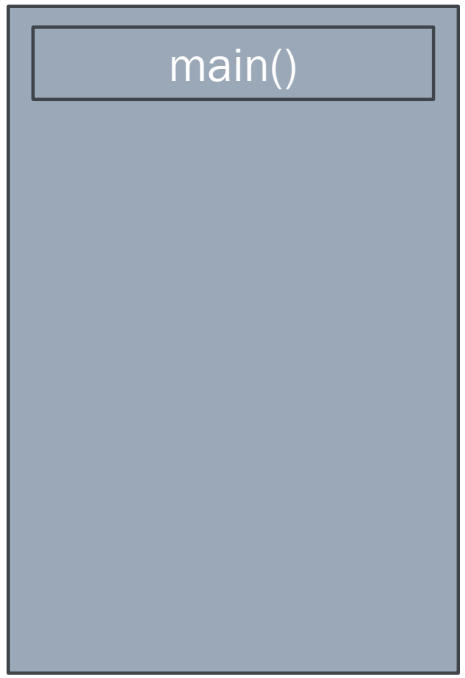
Stack-Speicher - Beispiel



Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;                    //Rückgabe
    }
}
```

Stack-Speicher - Beispiel

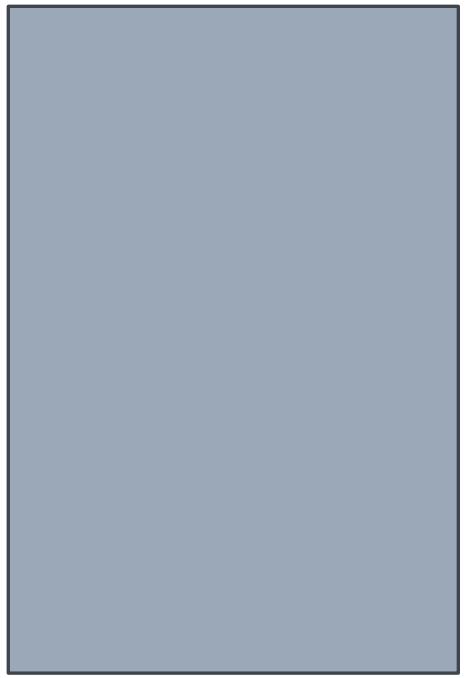


Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;
        int y = sum(x, 5);
        System.out.println(y);              //Konsolenausgabe
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

SMART INDUSTRY CAMPUS

Stack-Speicher - Beispiel



Stack

```
public class StackExample {
    public static void main(String[] args) {
        int x = 10;
        int y = sum(x, 5);
        System.out.println(y);
    }

    public static int sum(int a, int b) {
        return a + b;
    }
}
```

SMART INDUSTRY CAMPUS

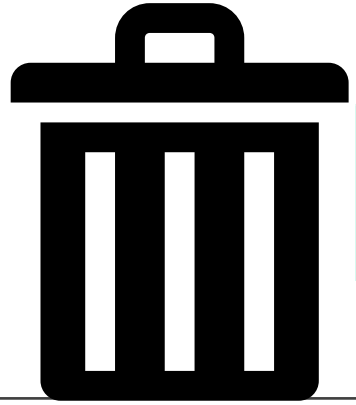
Heap-Speicher

- wird für Objekte und ihren Referenzvariablen (Klassen/Instanzvariablen) verwendet
- Objekt wird so lange gespeichert, bis keine Referenz mehr auf das Objekt zeigt
- Garbage Collector - Prozesse

SMART INDUSTRY CAMPUS

Garbage Collector

Garbage Collector



SMART INDUSTRY CAMPUS

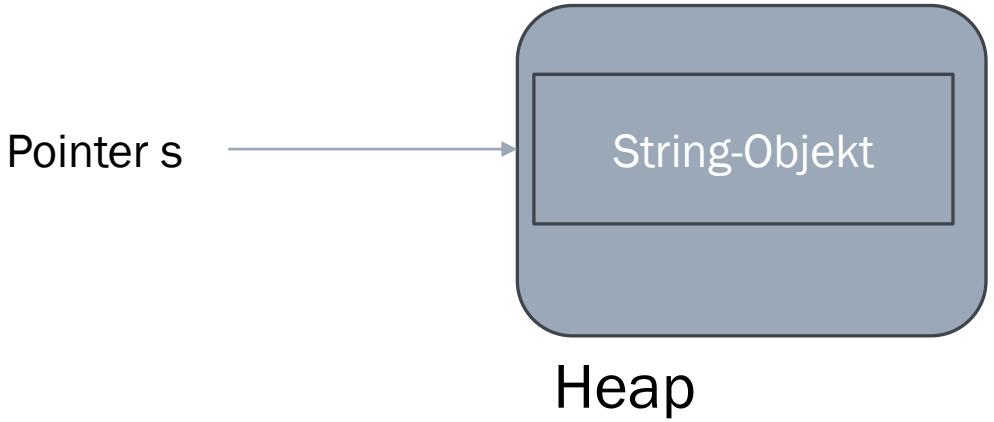
- wird von der JVM gesteuert
- je nach JVM-Implementation funktioniert der Garbage Collector unterschiedlich
- löscht automatisch nicht mehr erreichbare Objekte aus dem Heap
- Vokabel:
  - X Objects are eligible for the garbage collector
  - X Objekte kommen für den Garbage Collector in Frage

SMART INDUSTRY CAMPUS

Garbage Collector – Nulling Reference

```
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo"); //String-Objekt wird auf Heap und StringPool gespeichert
        s = null; //Nichts zeigt mehr auf das Objekt -> Objekt wird gelöscht
    }
}
```

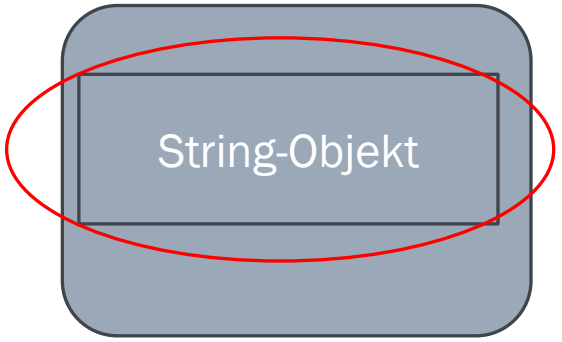
Nulling Reference




```java
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo");
        //s = null;
    }
}
```

Nulling Reference

Pointer s



Heap

EINS FÜR GARBARGE COLLECTOR ZUGÄNGLICH

```
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo");
        s = null;
    }
}
```

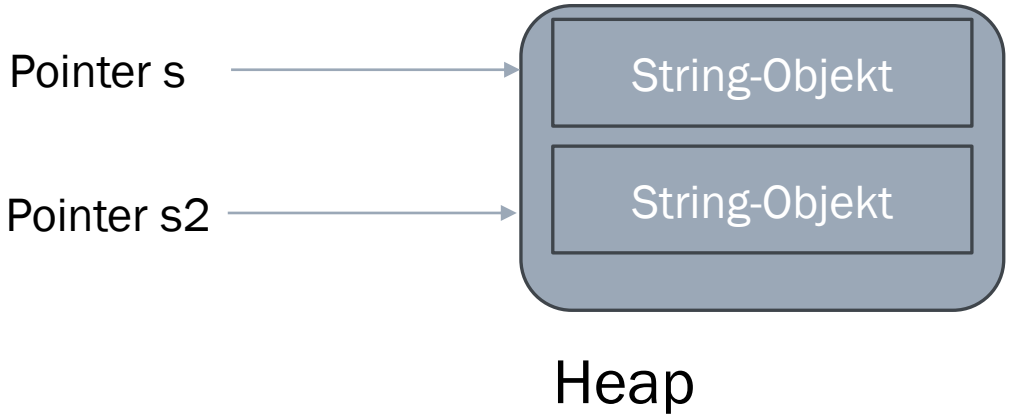
SMART INDUSTRY CAMPUS

Garbage Collector – Reassigning Reference

```
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo");
        String s2 = new String("Tschüss");
        s = s2;
    }
}
```

SMART INDUSTRY CAMPUS

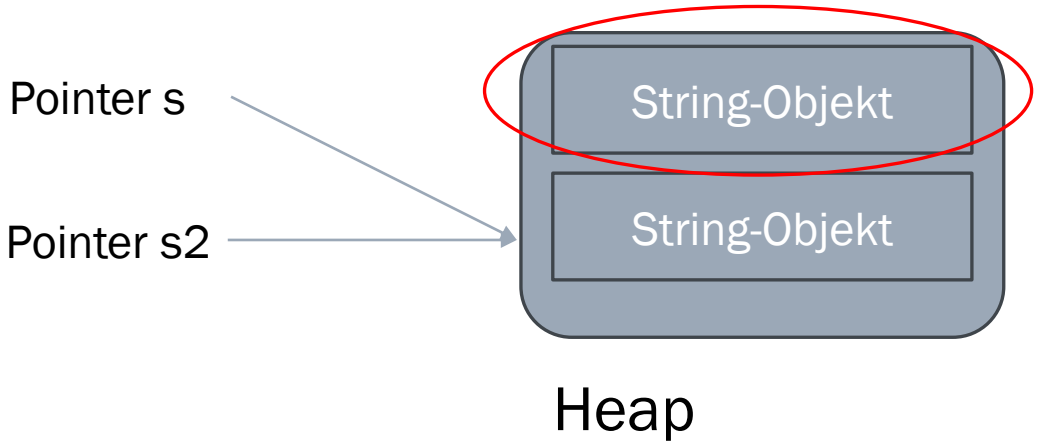
Garbage Collector – Reassigning Reference




```java
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo");
        String s2 = new String("Tschüss");
        //s = s2;
    }
}
```

SMART INDUSTRY CAMPUS

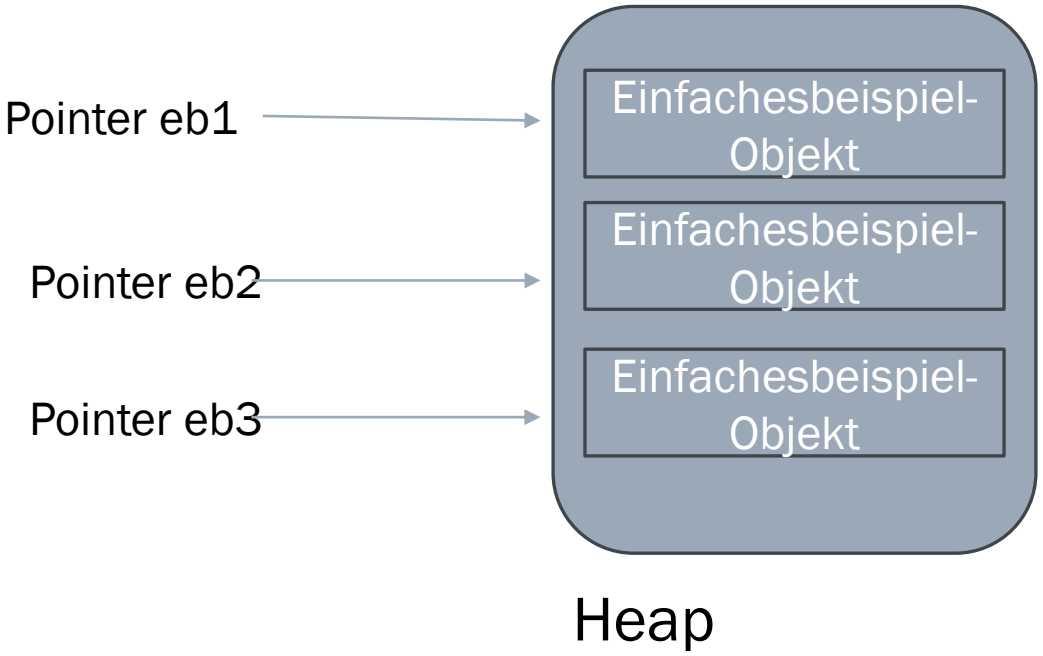
Garbage Collector – Reassigning Reference




```java
public class EinfachesBeispiel {
    public static void main(String[] args) {
        String s = new String("Hallo");
        String s2 = new String("Tschüss");
        s = s2;
    }
}
```

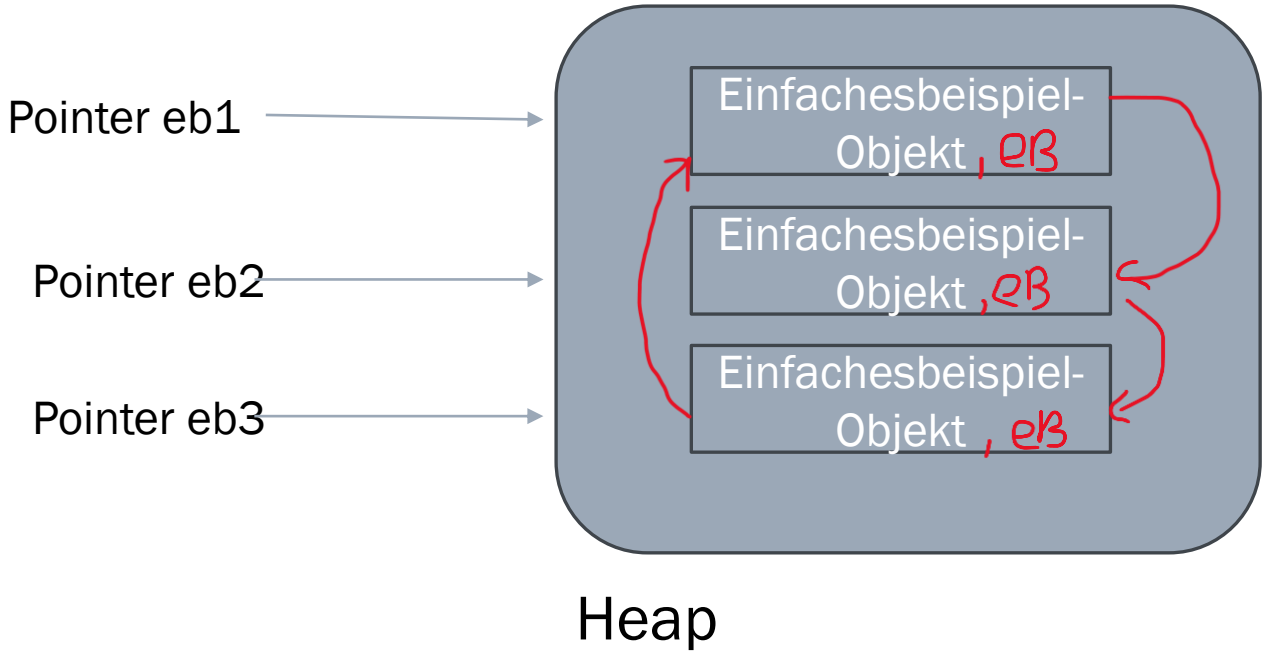
STRING-OBJEKT „Hallo“ FÜR GARBAGE COLLECTOR

Garbage Collector – Isolating Reference




```java
public class EinfachesBeispiel {
    EinfachesBeispiel eB;

    public static void main(String[] args) {
        EinfachesBeispiel eb1 = new EinfachesBeispiel();
        EinfachesBeispiel eb2 = new EinfachesBeispiel();
        EinfachesBeispiel eb3 = new EinfachesBeispiel();

        //eb1.eB = eb2;
        //eb2.eB = eb3;
        //eb3.eB = eb1;

        //eb1 = null;
        //eb2 = null;
        //eb3 = null;
    }
}
```

Garbage Collector – Isolating Reference




```java
public class EinfachesBeispiel {
    EinfachesBeispiel eB;

    public static void main(String[] args) {
        EinfachesBeispiel eb1 = new EinfachesBeispiel();
        EinfachesBeispiel eb2 = new EinfachesBeispiel();
        EinfachesBeispiel eb3 = new EinfachesBeispiel();

        eb1.eB = eb2;
        eb2.eB = eb3;
        eb3.eB = eb1;

        //eb1 = null;
        //eb2 = null;
        //eb3 = null;
    }
}
```

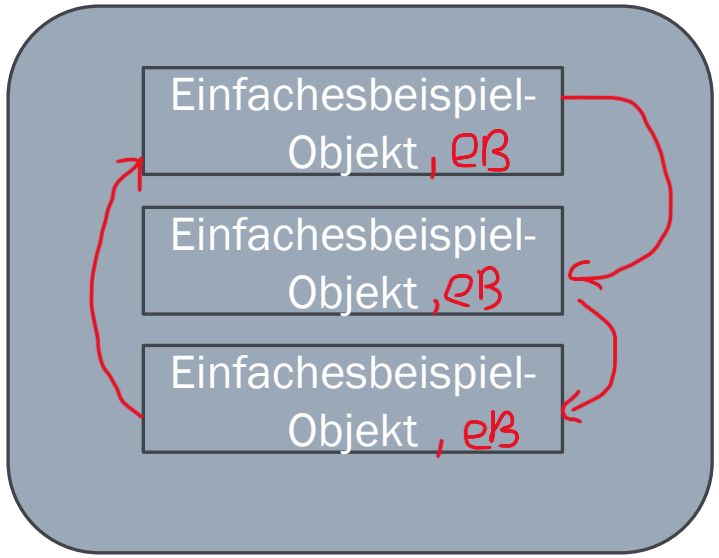
SMART INDUSTRY CAMPUS

Garbage Collector – Isolating Reference

Pointer eb1

Pointer eb2

Pointer eb3



Heap

ALLE 3 FÜR GARBAGE COLLECTOR

```
public class EinfachesBeispiel {
   EinfachesBeispiel eB;

   public static void main(String[] args) {
      EinfachesBeispiel eb1 = new EinfachesBeispiel();
      EinfachesBeispiel eb2 = new EinfachesBeispiel();
      EinfachesBeispiel eb3 = new EinfachesBeispiel();

      eb1.eB = eb2;
      eb2.eB = eb3;
      eb3.eB = eb1;

      eb1 = null;
      eb2 = null;
      eb3 = null;
   }
}
```

SMART INDUSTRY CAMPUS

Garbage Collector

- Objekte innerhalb einer Methode werden nach Abschluss der Methode weggeschmissen
- statische Variablen gehören der Klasse
  - Solange die Klasse lebt, sind statische Variablen zugreifbar
- Mittels System.gc() kann man den Garbage Collector manuell laufen lassen
  - **Hängt aber von der JVM-Implementation ab, was genau passiert**

SMART INDUSTRY CAMPUS

Finalize()

- Die Klasse Object hat eine Methode finalize()
- wird garantiert einmalig vor dem Lauf des Garbage Collectors ausgeführt
- damit kann versucht werden, dass ein Objekt nicht vom Garbage Collector gelöscht wird
- ab Java 9 veraltet (=depricated)

SMART INDUSTRY CAMPUS

Zusammenfassung

- Garbage Collector löscht Objekte, die er nicht mehr erreichen kann
  - Ein Objekt ist nicht mehr erreichbar, wenn keine Referenz mehr darauf zugreift
- isolierte Objekte werden vom GC gelöscht
- statische Variablen leben solange die Klasse existiert

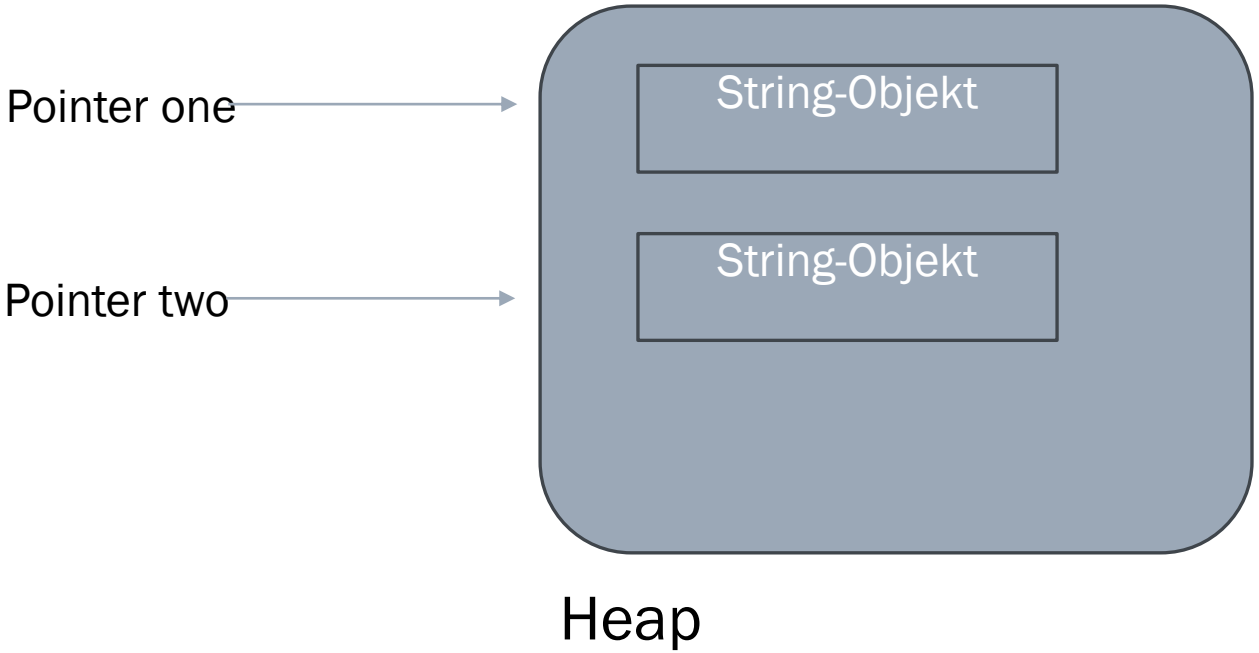
How many objects are eligible for garbage collection?

SMART INDUSTRY CAMPUS

- Garbage Collector löscht Objekte, die er nicht mehr erreichen kann
  - Ein Objekt ist nicht mehr erreichbar, wenn keine Referenz mehr darauf zugreift
- isolierte Objekte werden vom GC gelöscht
- statische Variablen leben solange die Klasse existiert

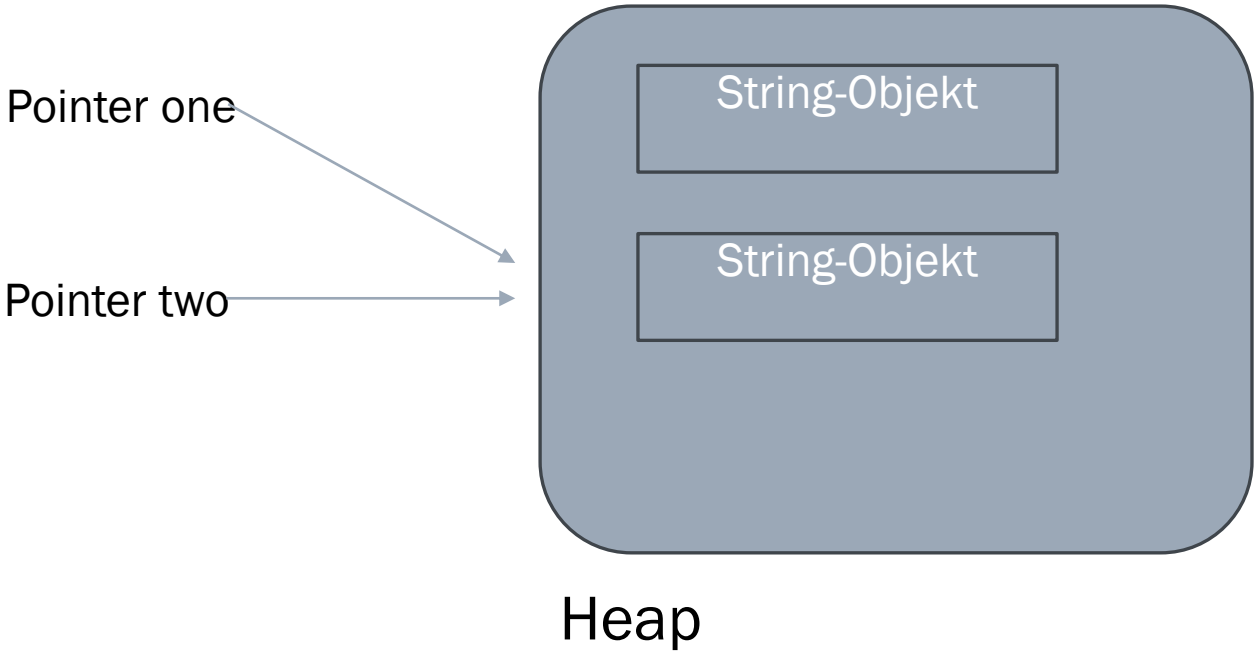
```
public class SCope {
   public static void main(String[] args) {
      String one,two;
      one = new String("a");
      two = new String("b");
      one = two;
      String three = one;
      one = null;
//HERE
   }
}
```

How many objects are eligible for garbage collection?



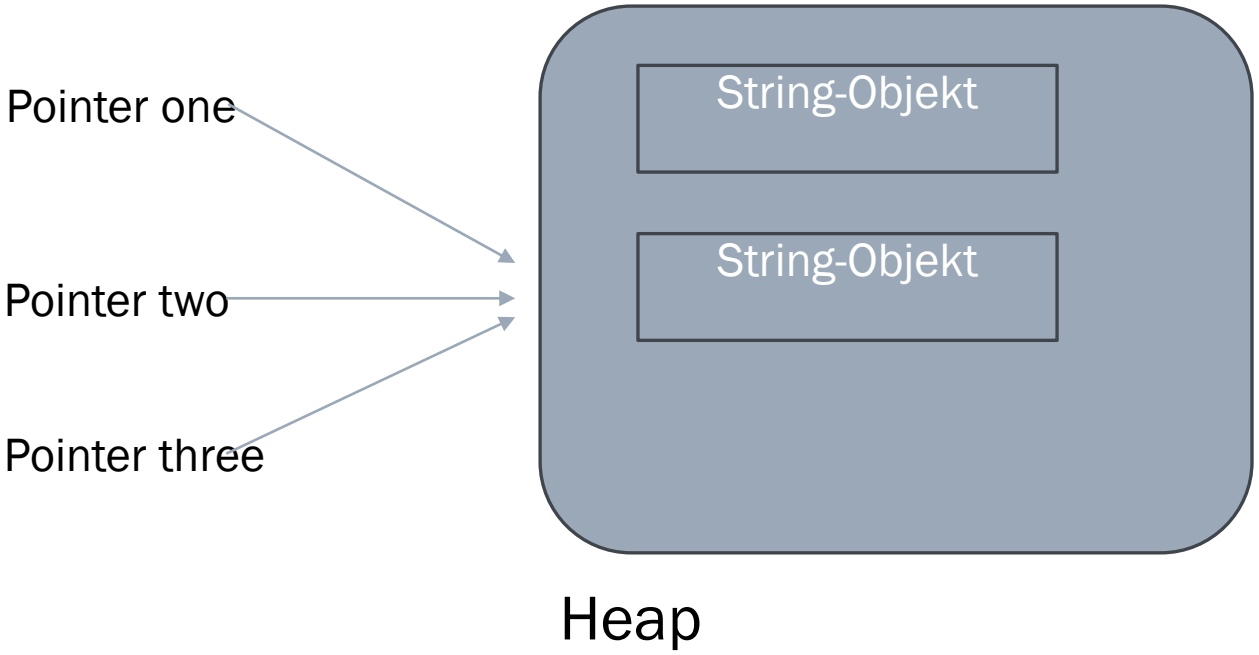

```
public class SCope {
   public static void main(String[]
args) {
      String one,two;
      one = new String("a");
      two = new String("b");
      //one = two;
      //String three = one;
      //one = null;
   }
}
```

How many objects are eligible for garbage collection?



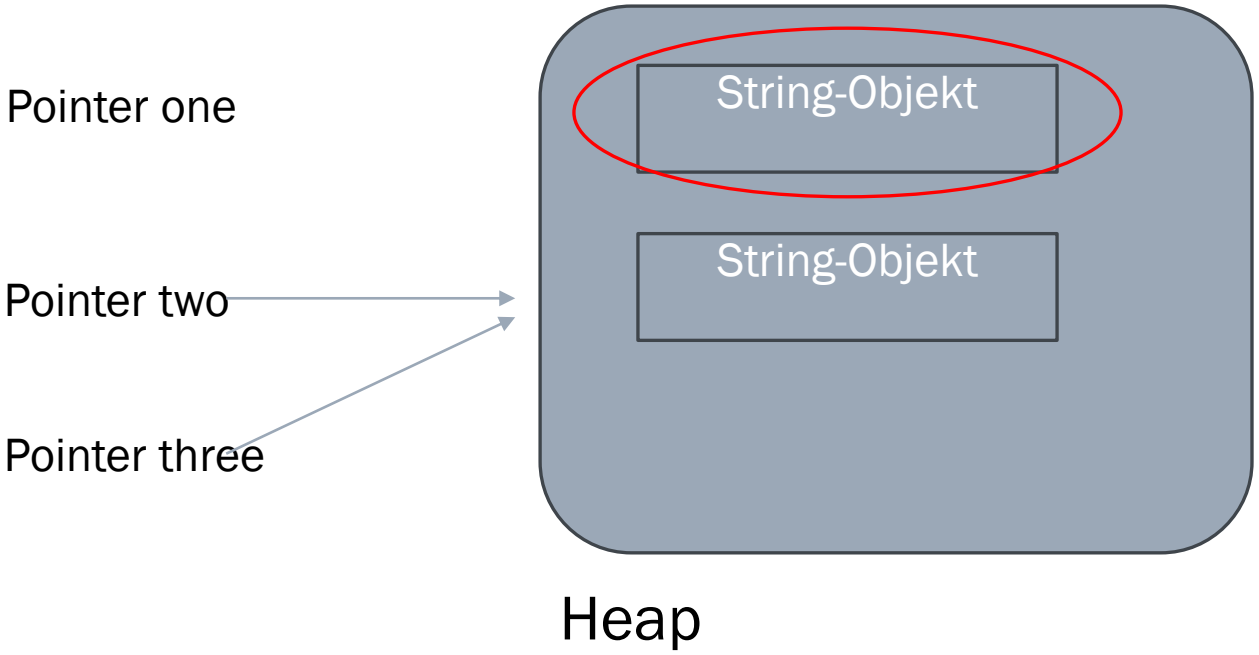

```
public class SCope {
   public static void main(String[]
args) {
      String one,two;
      one = new String("a");
      two = new String("b");
      one = two;
      //String three = one;
      //one = null;
   }
}
```

How many objects are eligible for garbage collection?




```
public class SCope {
   public static void main(String[]
args) {
      String one,two;
      one = new String("a");
      two = new String("b");
      one = two;
      String three = one;
      //one = null;
   }
}
```

How many objects are eligible for garbage collection?



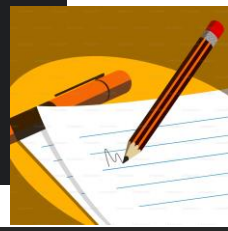

```
public class SCope {
   public static void main(String[]
args) {
      String one,two;
      one = new String("a");
      two = new String("b");
      one = two;
      String three = one;
      one = null;
//HERE
   }
}
```

How many objects are eligible for garbage collection?

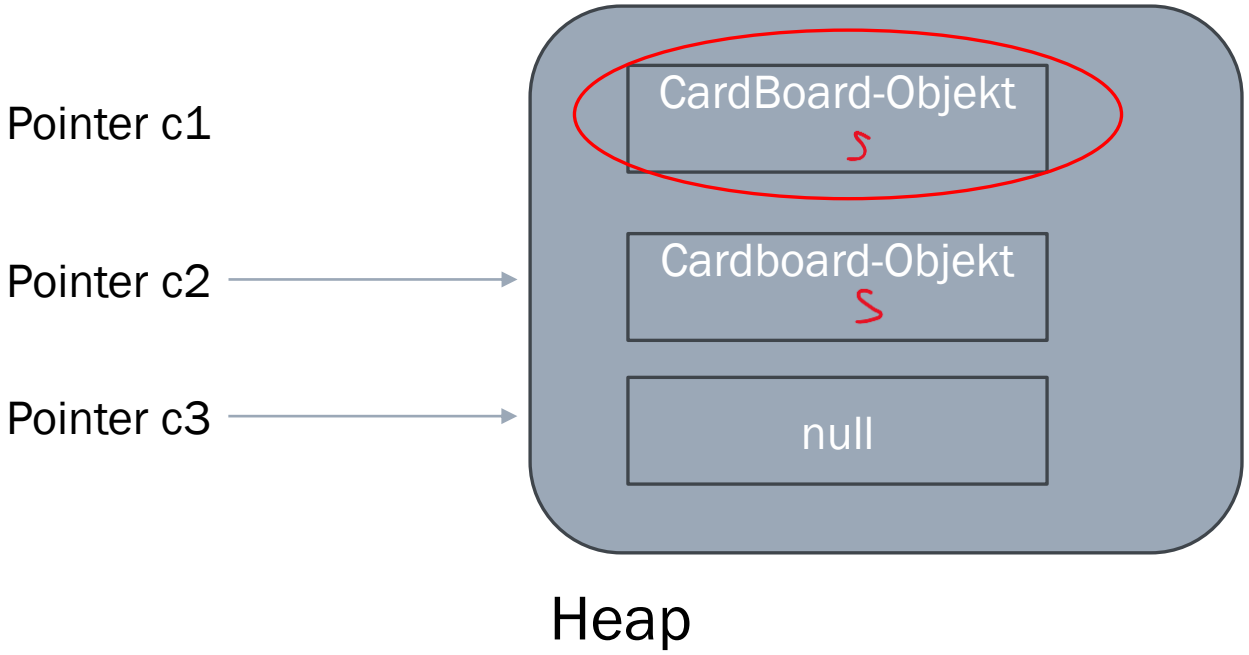
SMART INDUSTRY CAMPUS

- Garbage Collector löscht Objekte, die er nicht mehr erreichen kann
  - Ein Objekt ist nicht mehr erreichbar, wenn keine Referenz mehr darauf zugreift
- isolierte Objekte werden vom GC gelöscht
- statische Variablen leben solange die Klasse existiert

```
public class CardBoard {
   Short s = 200;
   CardBoard go(CardBoard cb){
      cb = null;
      return cb;
   }

   public static void main(String[] args) {
      CardBoard c1 = new CardBoard();
      CardBoard c2 = new CardBoard();
      CardBoard c3 = c1.go(c2);
      c1 = null;
//HERE
   }
}
```

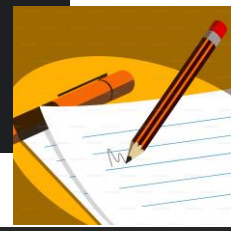


How many objects are eligible for garbage collection?




```
public class CardBoard {
   Short s = 200;
   CardBoard go(CardBoard cb){
      cb = null;
      return cb;
   }

   public static void main(String[] args) {
      CardBoard c1 = new CardBoard();
      CardBoard c2 = new CardBoard();
      CardBoard c3 = c1.go(c2);
      c1 = null;
   }
}
```

CardBoard-Objekt mit dem dazugehörigen Short-Objekt = 2 Objekte



SMART INDUSTRY CAMPUS

How many objects are eligible for garbage collection, if the code reaches //HERE?

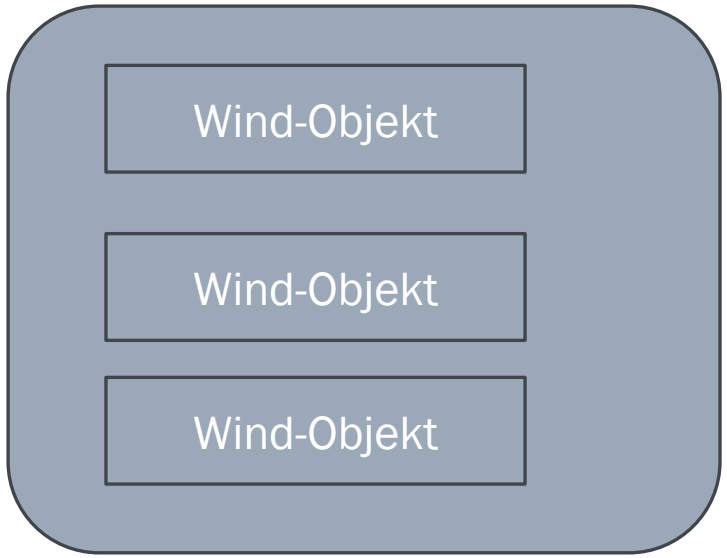
- Garbage Collector löscht Objekte, die er nicht mehr erreichen kann
  - Ein Objekt ist nicht mehr erreichbar, wenn keine Referenz mehr darauf zugreift
- isolierte Objekte werden vom GC gelöscht
- statische Variablen leben solange die Klasse existiert

```
public class Wind {
   int id;
   Wind(int id){
      this.id = id;
   }

   public static void main(String[] args) {
      new Wind(3).go();
//HERE
   }

   void go(){
      Wind w1 = new Wind(1);
      Wind w2 = new Wind(2);
      System.out.println(w1.id + " " + w2.id);
   }
}
```

How many objects are eligible for garbage collection, if the code reaches //HERE?

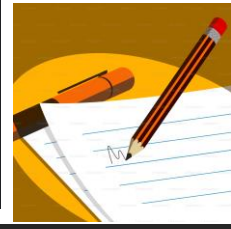
Pointer w1

Pointer w2



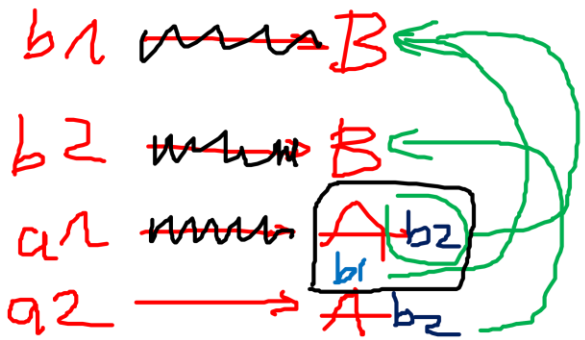
Heap

```
public class Wind {
   int id;
   Wind(int id){
      this.id = id;
   }

   public static void main(String[] args) {
      new Wind(3).go();
//HERE
   }

   void go(){
      Wind w1 = new Wind(1);
      Wind w2 = new Wind(2);
      System.out.println(w1.id + " " + w2.id);
   }
}
```



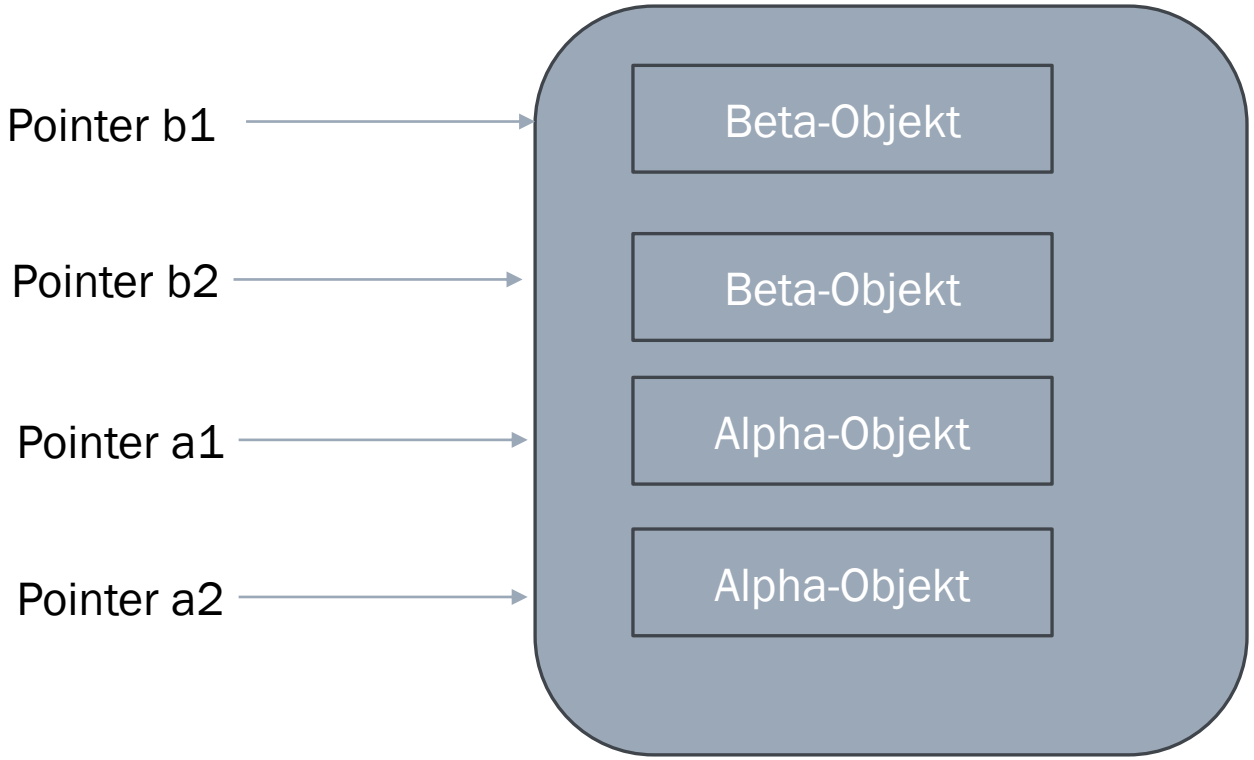
How many objects are eligible for garbage collection, if the code reaches //HERE?

- Garbage Collector löscht Objekte, die er nicht mehr erreichen kann
  - Ein Objekt ist nicht mehr erreichbar, wenn keine Referenz mehr darauf zugreift
- isolierte Objekte werden vom GC gelöscht
- statische Variablen leben solange die Klasse existiert



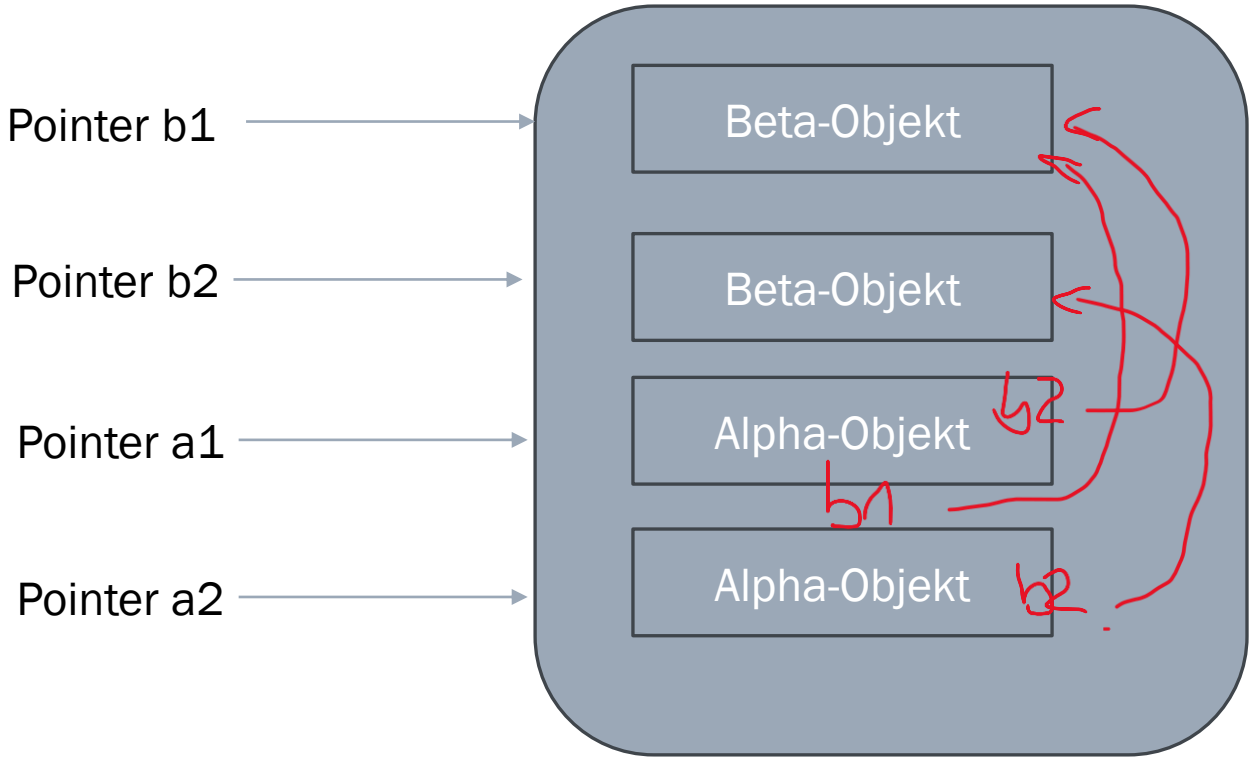

```
class Beta{ }
class Alpha{
   static Beta b1;
   Beta b2;
}
public class StaticObject {
   public static void main(String[] args) {
      Beta b1 = new Beta();
      Beta b2 = new Beta();
      Alpha a1 = new Alpha();
      Alpha a2 = new Alpha();

      a1.b1 = b1;
      a1.b2 = b1;
      a2.b2 = b2;

      a1 = null; b1 = null; b2 = null;
      //HERE
   }
}
```

How many objects are eligible for garbage collection, if the code reaches //HERE?




```
class Beta{ }
class Alpha{
   static Beta b1;
   Beta b2;
}
public class StaticObject {
   public static void main(String[] args) {
      Beta b1 = new Beta();
      Beta b2 = new Beta();
      Alpha a1 = new Alpha();
      Alpha a2 = new Alpha();

      //a1.b1 = b1;
      //a1.b2 = b1;
      // a2.b2 = b2;

      //a1 = null; b1 = null; b2 = null;
      //HERE
   }
}
```

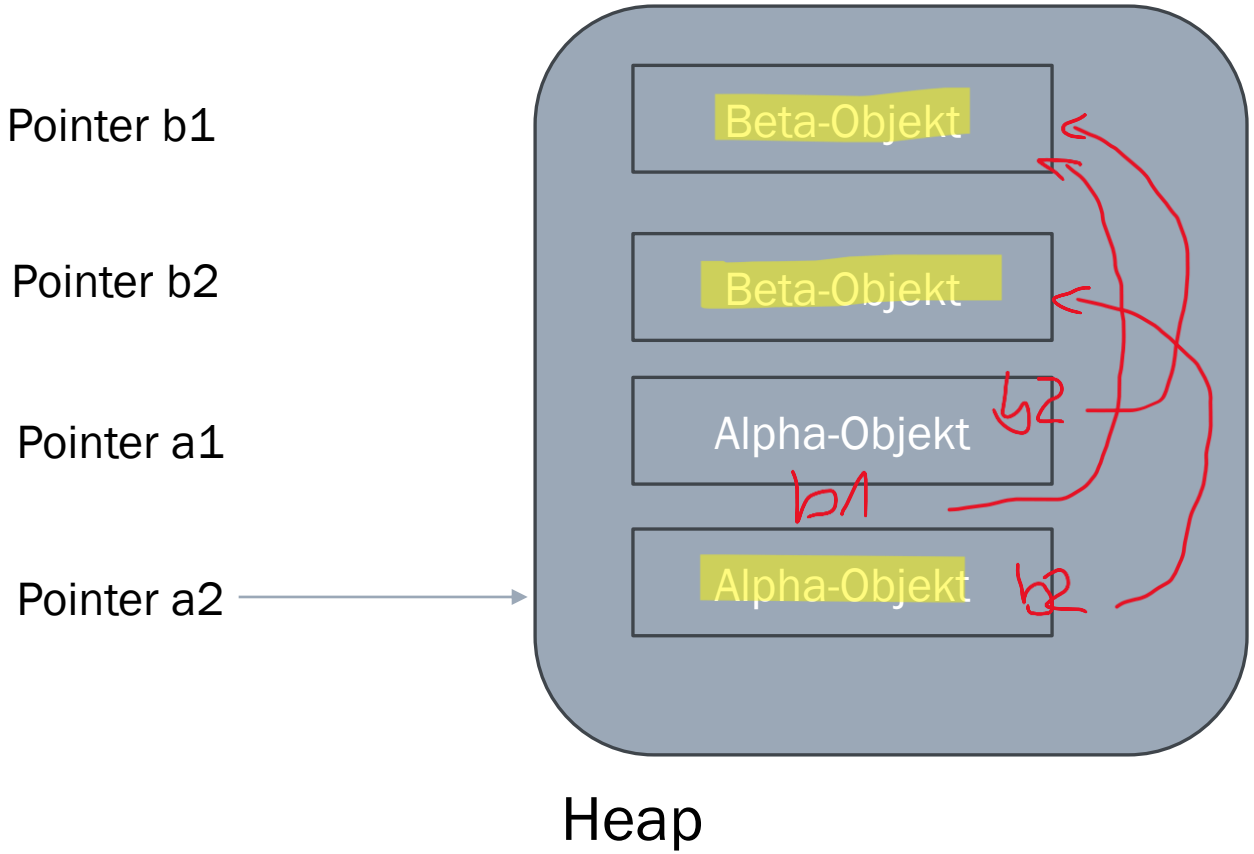
How many objects are eligible for garbage collection, if the code reaches //HERE?



Heap

```
class Beta{ }
class Alpha{
   static Beta b1;
   Beta b2;
}
public class StaticObject {
   public static void main(String[] args) {
      Beta b1 = new Beta();
      Beta b2 = new Beta();
      Alpha a1 = new Alpha();
      Alpha a2 = new Alpha();

      a1.b1 = b1;
      a1.b2 = b1;
      a2.b2 = b2;

      //a1 = null; b1 = null; b2 = null;
      //HERE
   }
}
```

How many objects are eligible for garbage collection, if the code reaches //HERE?




```
class Beta{ }
class Alpha{
   static Beta b1;
   Beta b2;
}
public class StaticObject {
   public static void main(String[] args) {
      Beta b1 = new Beta();
      Beta b2 = new Beta();
      Alpha a1 = new Alpha();
      Alpha a2 = new Alpha();

      a1.b1 = b1;
      a1.b2 = b1;
      a2.b2 = b2;

       a1 = null; b1 = null; b2 = null;
      //HERE
   }
}
```