OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

07 – Java Data Types (Часть 3)

Дмитрий Коган

Как меня слышно и видно?

Если нет – напишите, если слышите – смайлик в чат.

Цели :

- Научимся создавать и инициализировать Java-объекты
- Изучим жизненный цикл Java-объекта
- Понаблюдаем за гибелью объектов

Начинаем?

Темы экзамена

- ❑ Java Basics
- ❑ Working with Java Data Types
- ❑ Using Operators and Decision Constructs
- ❑ Creating and Using Arrays
- ❑ Using Loop Constructs
- ❑ Working with Methods and Encapsulation
- ❑ Working with Inheritance
- ❑ Handling Exceptions
- ❑ Working with Selected classes from the Java API

Подтемы экзамена

Working with Java Data Types

- Declare and initialize variables (including casting of primitive data types)
- Differentiate between object reference variables and primitive variables
- Know how to read or write to object fields
- Explain an object's lifecycle (creation, "dereference by reassignment" and garbage collection)
- Develop code that uses wrapper classes such as Boolean, Double and Integer

Создание объекта

Конструктор

```
class Person {}
class ObjectLifeCycle {
    Person person;
}
```

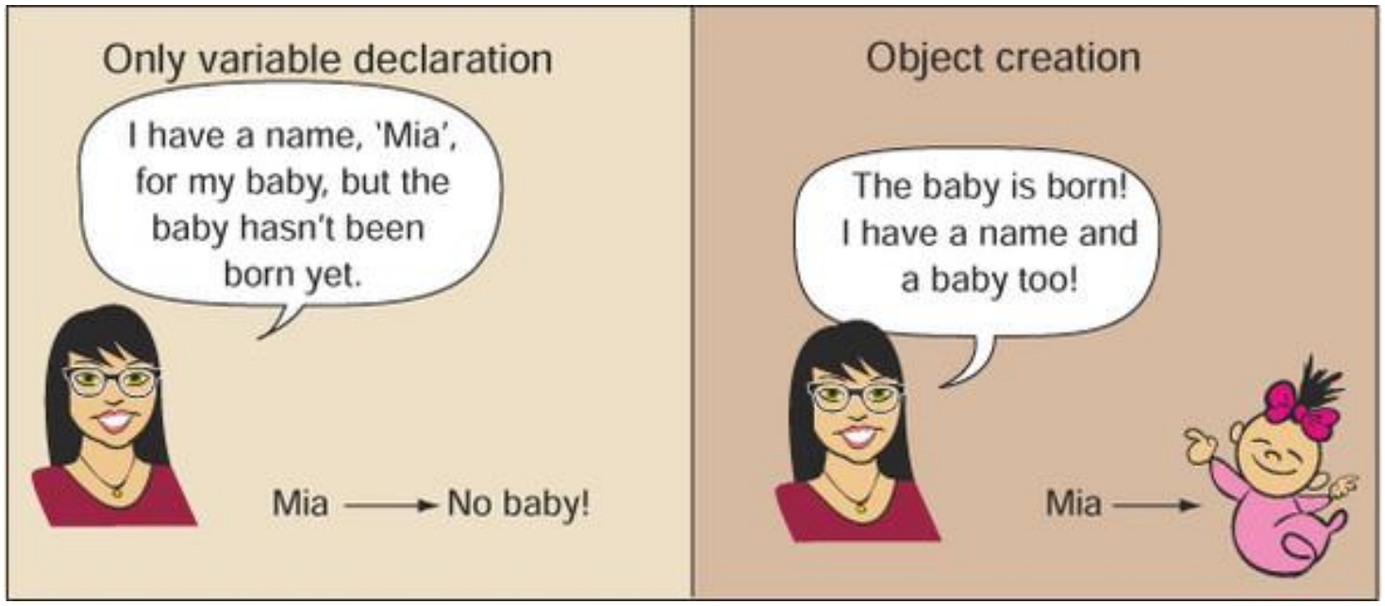
Class Person

Declaring a reference variable of type Person

```
class ObjectLifeCycle2 {
    Person person = new Person();
}
```

Declaring and initializing a variable of type Person

Конструктор




```
class ObjectLifeCycle2 {
    Person person = new Person();
    ObjectLifeCycle2() {
        new Person();
    }
}
```

An unreferenced object

Строки

```
class ObjectLifeCycle3 {
    String obj1 = new String("eJava");
    String obj2 = "Guru";
}
```

String object referenced by obj1

Another String object referenced by obj2

Конструктор или нет?

```
public class Chick {
  public Chick() {
    System.out.println("in constructor");
  }
}

public void Chick() { } // NOT A CONSTRUCTOR
```

Инициализация

```
public class Chicken {
  int numEggs = 0;// initialize on line
  String name;
  public Chicken() {
    name = "Duke";// initialize in constructor
  } }
```

Упражнение

```
public class ExamTaker {
    private String name;
    private int score;
    private String exam;
    private boolean passed;
    ExamTaker(){}
    ExamTaker(String name, int score, String exam){
        this.name = name;
        this.score = score;
        this.exam = exam;
    }
    public void show() {
        System.out.println("Name: " + name + ", passed: " + passed);
    }
    public boolean isPassed() {
        return passed = score >= 65;
    }
    public static void main(String[] args) {
        // line X
    }
}
```

Which statement, when inserted at line X, correctly initializes an ExamTaker instance?

A. ExamTaker examTaker = "Bob";
B. ExamTaker examTaker = ExamTaker.new("Bob", 100, "1Z0-808");
C. ExamTaker examTaker = new ExamTaker();
D. ExamTaker examTaker = ExamTaker("Bob", 100, "1Z0-808");
E. None of the above

Ответ: C

Запись и чтение

```
public class Swan {
  int numberEggs;// instance variable
  public static void main(String[] args) {
    Swan mother = new Swan();
    mother.numberEggs = 1;    // set variable
    System.out.println(mother.numberEggs);  // read variable
  }
}
```

```
1: public class Name {
2:   String first = "Theodore";
3:   String last = "Moose";
4:   String full = first + last;
5: {
```

Блоки инициализации

```
3: public static void main(String[] args) {
4:    { System.out.println("Feathers"); }
5: }
6: { System.out.println("Snowy"); }
```

Порядок инициализации

- ✓ Поля и блоки инициализации объекта отрабатывают в том порядке, в котором они расположены в файле.
- ✓ Только после них отрабатывают конструкторы.

Что выведется?

```
1: public class Chick {
2:  private String name = "Fluffy";
3:  { System.out.println("setting field"); }
4:  public Chick() {
5:    name = "Tiny";
6:    System.out.println("setting constructor");
7:  }
8:   public static void main(String[] args) {
9:     Chick chick = new Chick();
10:   System.out.println(chick.name); } }
```

Порядок инициализации

```
1: public class Chick {
2:  private String name = "Fluffy";
3:  { System.out.println("setting field"); }
4:  public Chick() {
5:    name = "Tiny";
6:    System.out.println("setting constructor");
7:  }
8:   public static void main(String[] args) {
9:     Chick chick = new Chick();
10:   System.out.println(chick.name); } }
```

Running this example prints this:

```
setting field
setting constructor
Tiny
```

Порядок важен

```
{ System.out.println(name); }  // DOES NOT COMPILE
private String name = "Fluffy";
```

Упражнение

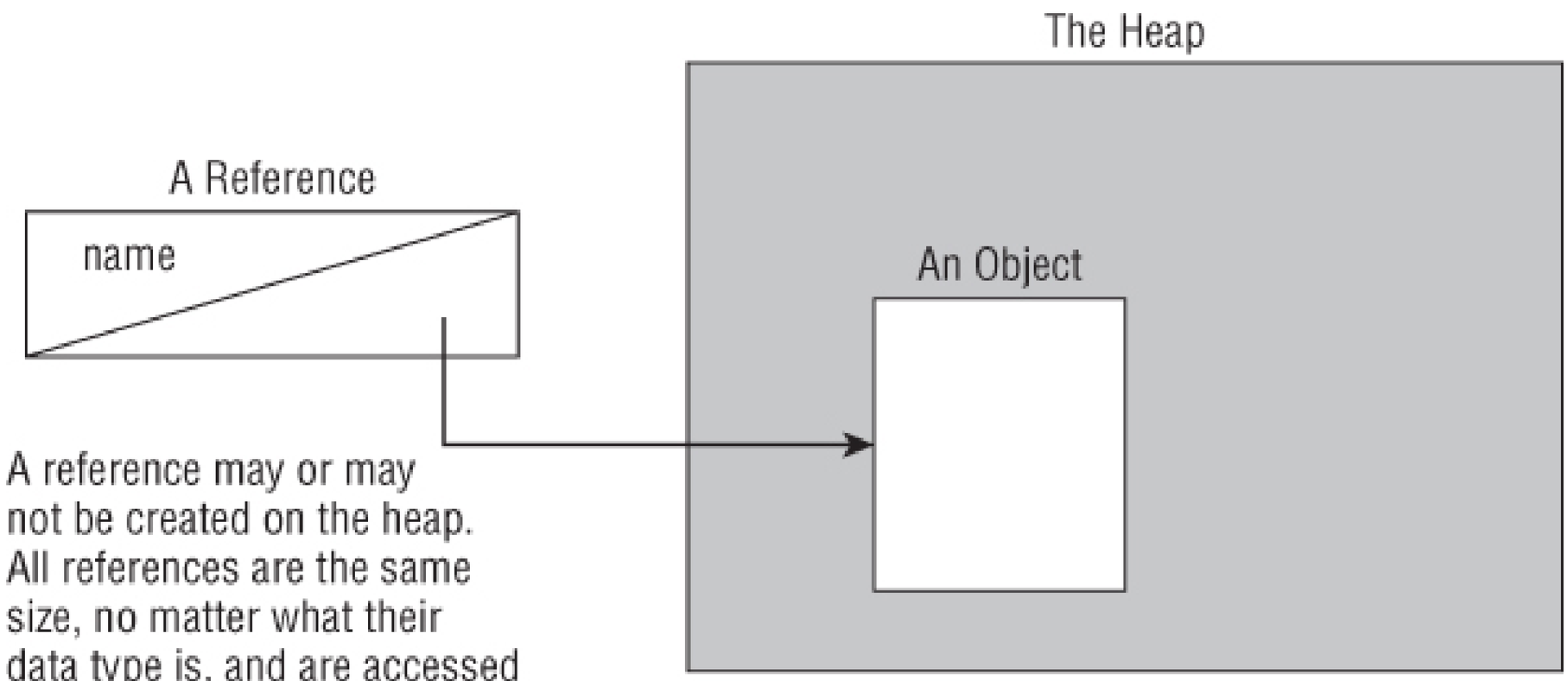
```
public class Egg {
  public Egg() {
    number = 5;
  }
  public static void main(String[] args) {
    Egg egg = new Egg();
    System.out.println(egg.number);
  }
  private int number = 3;
  { number = 4; } }
```

Ответ: 5

Вопросы?

Доступность объектов

Объекты и ссылки



A reference may or may not be created on the heap. All references are the same size, no matter what their data type is, and are accessed by their variable name.

Objects are always on the heap. They have no name and can only be accessed via a reference. Objects vary in size depending on their class definition.

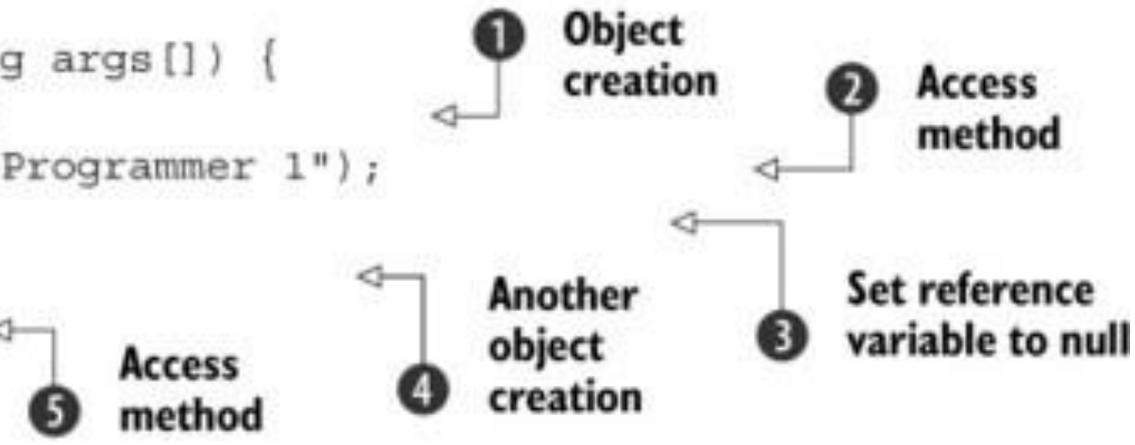
Объект доступен

- ✓ Чтобы сделать поля и методы «живого» объекта доступными для других объектов, мы даем на него ссылку при помощи ссылочной переменной.
- ✓ Простое объявление ссылочной переменной вовсе не означает, что будет создан новый объект.

Объект доступен

```
class Exam {
    String name;
    public void setName(String newName) {
        name = newName;
    }
}
```

```
class ObjectLife1 {
    public static void main(String args[]) {
        Exam myExam = new Exam();
        myExam.setName("OCA Java Programmer 1");
        myExam = null;
        myExam = new Exam();
        myExam.setName("PHP");
    }
}
```



Упражнение

```
public class ObjectsGalore {
    static ObjectsGalore obj1, obj2, obj3, obj4;
    int numberOfObjects;
    public static int addObject(ObjectsGalore obj4) {
        return obj4.numberOfObjects *= 2;
    }
    public static void main(String[] args) {
        obj1 = obj2 = obj3 = new ObjectsGalore();
        obj4 = obj2;
        addObject(obj4);
    }
}
```

How many objects of the class ObjectsGalore are created in memory at run time?

A. 1
B. 4
C. 8
D. Compilations fails

Ответ: А

Объект недоступен

VARIABLE GOES OUT OF SCOPE

```
public void myMethod() {
    int result = 88;
    if (result > 78) {
        Exam myExam1 = new Exam();
        myExam1.setName("Android");
    }
    else {
        Exam myExam2 = new Exam();
        myExam2.setName("MySQL");
    }
}
```

1. Scope of local variable myExam1
2. Start of else block
3. End of else block

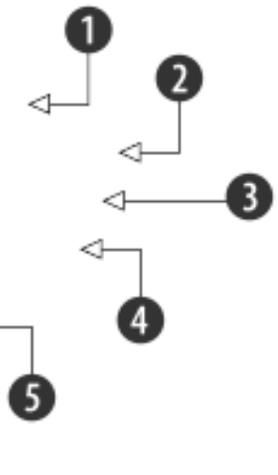
Объект недоступен

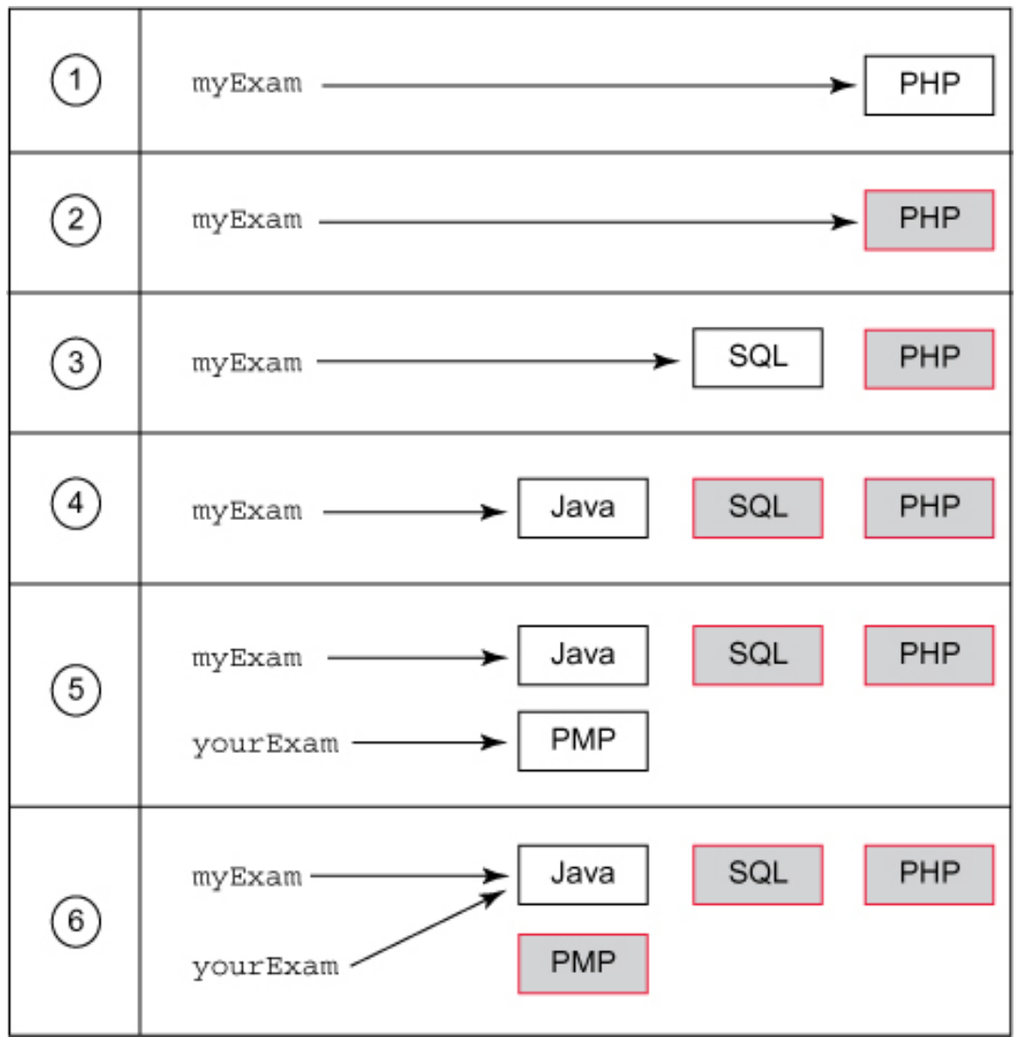
DEREFERENCING BY REASSIGNMENT

```
class Exam {
    String name;
    public Exam(String name) {
        this.name = name;
    }
}
class ObjectLife2 {
    public static void main(String args[]) {
        Exam myExam = new Exam("PHP");
        myExam = null;
        myExam = new Exam("SQL");
        myExam = new Exam("Java");

        Exam yourExam = new Exam("PMP");
        yourExam = myExam;
    }
}
```

Объект недоступен

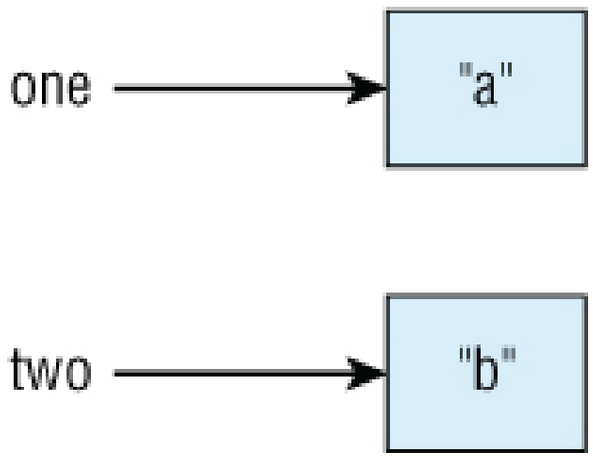
```
class ObjectLife2 {
    public static void main(String args[]) {
        Exam myExam = new Exam("PHP");
        myExam = null;
        myExam = new Exam("SQL");
        myExam = new Exam("Java");

        Exam yourExam = new Exam("PMP");
        yourExam = myExam;
    }
}
```

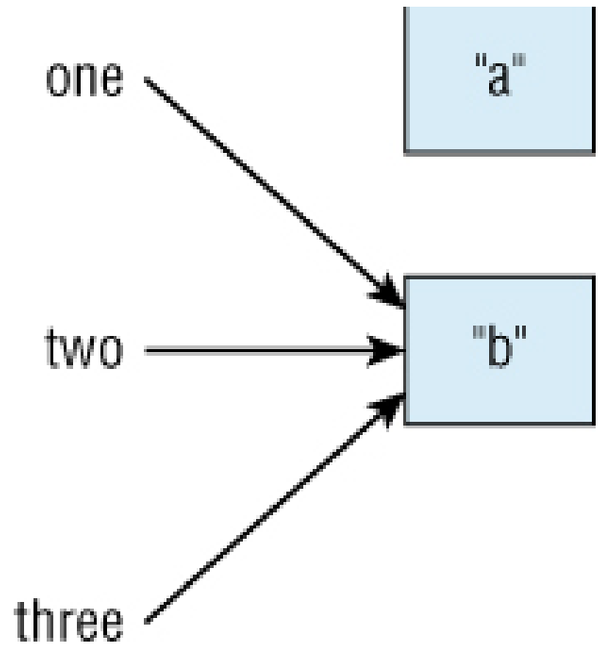




Рисуем стрелочки

```
1: public class Scope {
2:  public static void main(String[] args) {
3:    String one, two;
4:    one = new String("a");
5:    two = new String("b");
6:    one = two;
7:    String three = one;
8:    one = null;
9:  } }
```





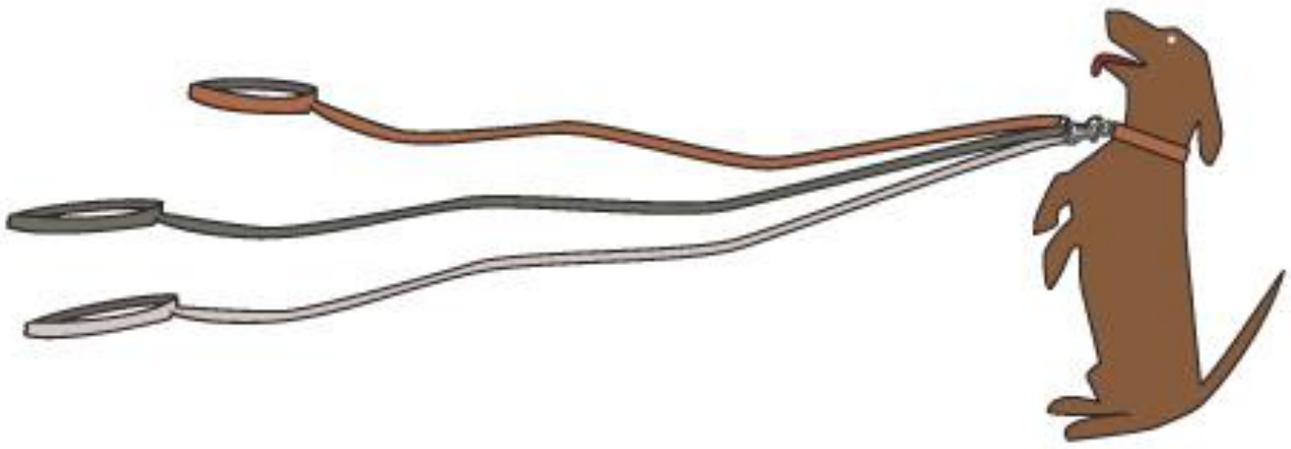
Вопросы?

Сборка мусора

Ветеринарный контроль

A leash without a dog.

A dog without a leash.



Several leashes may be tethered to one dog.

Авто-помощник

- ✓ Недоступные объекты удаляются благодаря механизму т.н. сборки мусора. Цель сборки мусора состоит в том, чтобы выявлять и удалять объекты, которые больше не нужны нашему приложению, чтобы можно было высвободить и заново использовать системные ресурсы.

Недоступные объекты

- ✓ Объект становится недоступным, если на него больше не указывает ни одна ссылка; это может случиться по следующим причинам:
  – ссылочную переменную «перекинули» на другой объект,
  – ссылочной переменной присвоили значение null, или
  – закончилась область действия данной ссылочной переменной.

Обнуление ссылки

```
1. public class GarbageTruck {
2.   public static void main(String [] args) {
3.     StringBuffer sb = new StringBuffer("hello");
4.     System.out.println(sb);
5.     // The StringBuffer object is not eligible for collection
6.     sb = null;
7.     // Now the StringBuffer object is eligible for collection
8.   }
9. }
```

Ссылка переопределена

```
class GarbageTruck {
  public static void main(String [] args) {
    StringBuffer s1 = new StringBuffer("hello");
    StringBuffer s2 = new StringBuffer("goodbye");
    System.out.println(s1);
    // At this point the StringBuffer "hello" is not eligible
    s1 = s2; // Redirects s1 to refer to the "goodbye" object
    // Now the StringBuffer "hello" is eligible for collection
}}
```

Объект спасён

```
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;
  }
}
```

Изолированный остров

```
class Exam {
    private String name;
    private Exam other;
    public Exam(String name) {
        this.name = name;
    }
    public void setExam(Exam exam) {
        other = exam;
    }
}
class IslandOfIsolation {
    public static void main(String args[]) {
        Exam php = new Exam("PHP");
        Exam java = new Exam("Java");

        php.setExam(java);
        java.setExam(php);

        php = null;
        java = null;
    }
}
```

1. Initialize variable php
2. Initialize variable java
3. Assign object referred by java to php.exam
4. Assign object referred by php to java.exam
5. Assign null to php
6. Assign null to java

Изолированный остров

```
class Exam {
    private String name;
    private Exam other;
    public Exam(String name) {
        this.name = name;
    }
    public void setExam(Exam exam) {
        other = exam;
    }
}
class IslandOfIsolation {
    public static void main(String args[]) {
        Exam php = new Exam("PHP");
        Exam java = new Exam("Java");

        php.setExam(java);
        java.setExam(php);

        php = null;
        java = null;
    }
}
```

❶ **Initialize variable php**

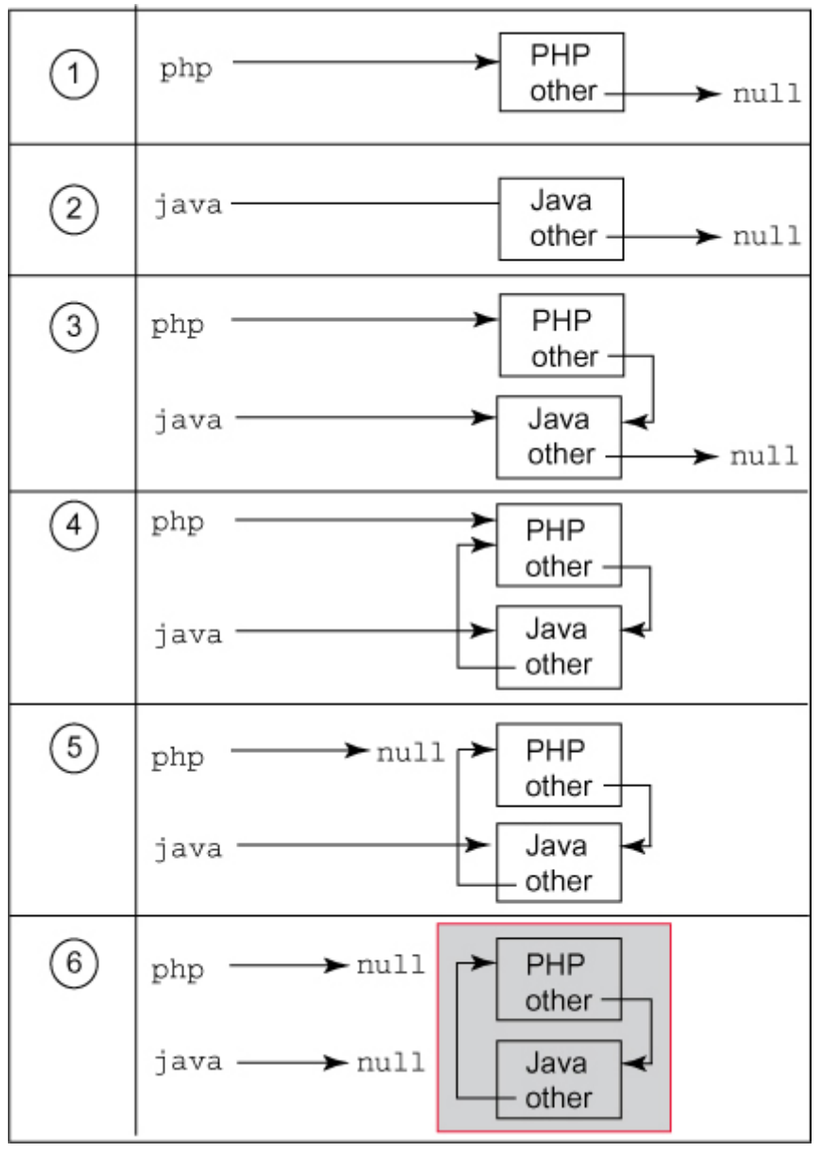
❷ **Initialize variable java**

❸ **Assign object referred by java to php.exam**

❹ **Assign object referred by php to java.exam**

❺ **Assign null to php**

❻ **Assign null to java**

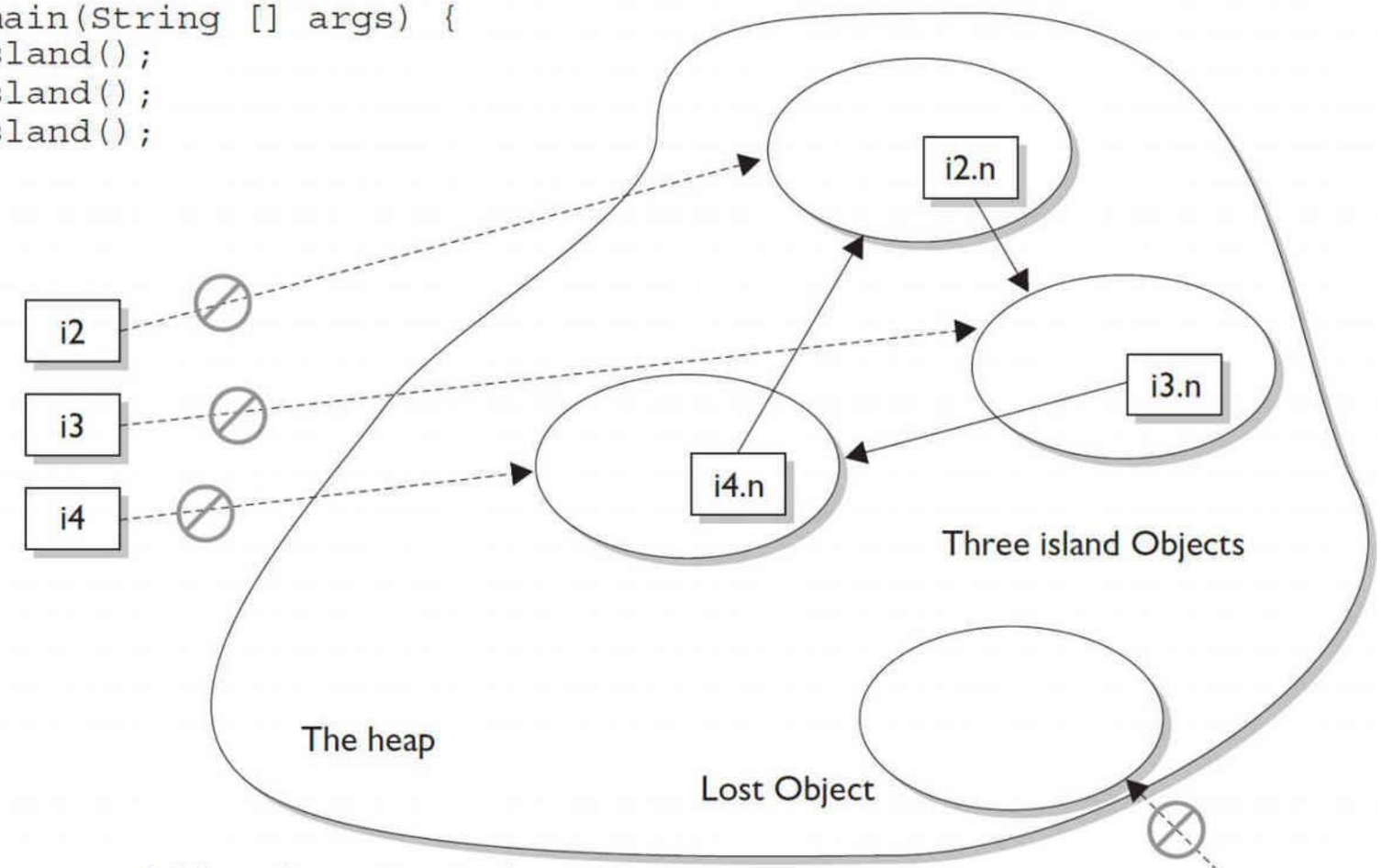


Изолированный остров

```
public class Island {
  Island i;
  public static void main(String [] args) {

    Island i2 = new Island();
    Island i3 = new Island();
    Island i4 = new Island();

    i2.i = i3;   // i2 refers to i3
    i3.i = i4;   // i3 refers to i4
    i4.i = i2;   // i4 refers to i2

    i2 = null;
    i3 = null;
    i4 = null;

    // do complicated, memory intensive stuff
  }
}
```

Изолированный остров

```
public class Island (
  Island n;
  public static void main(String [] args) {
    Island i2 = new Island();
    Island i3 = new Island();
    Island i4 = new Island();
    i2.n = i3;
    i3.n = i4;
    i4.n = i2;
    i2 = null;
    i3 = null;
    i4 = null;
    doComplexStuff();
  }
}
```



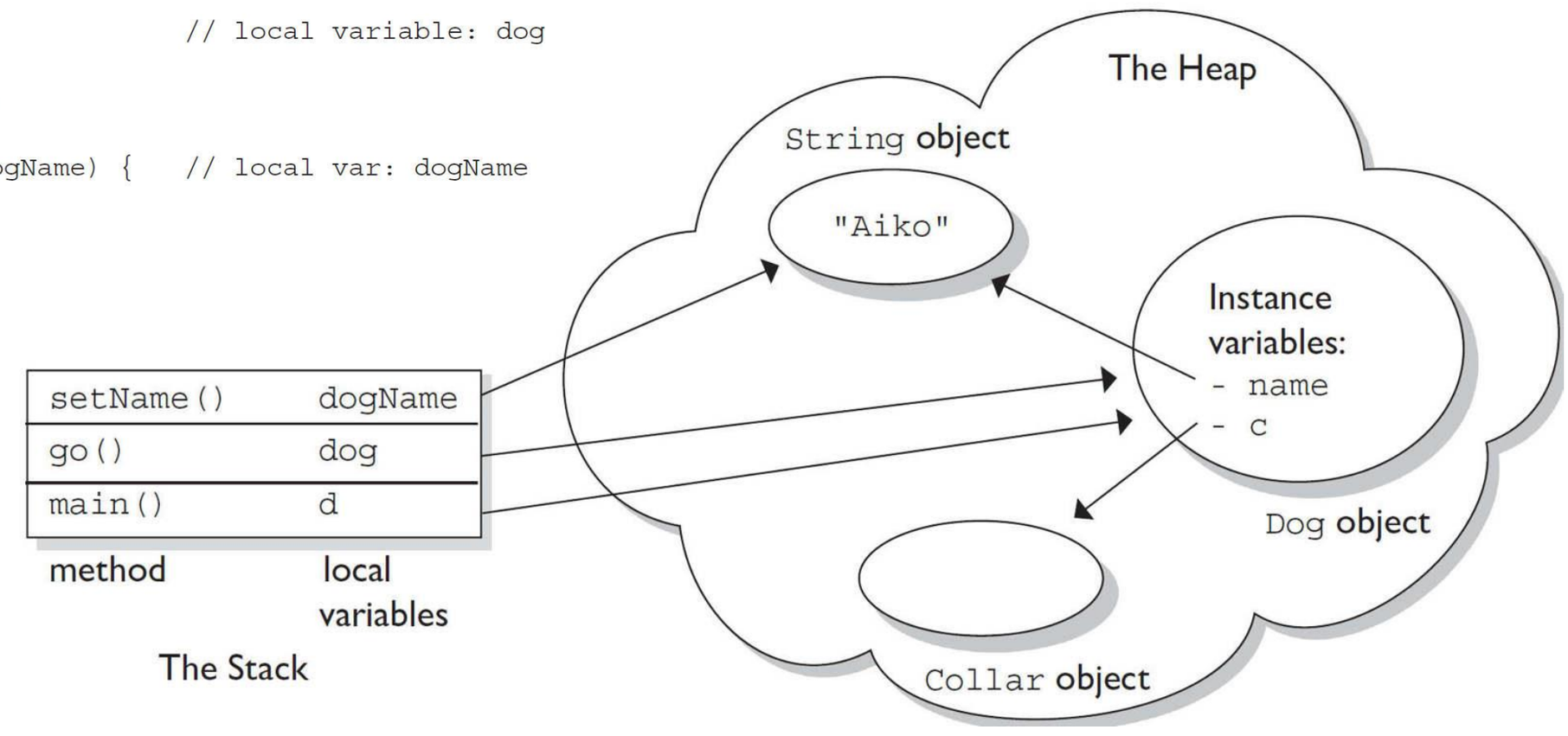
Indicated an active reference

Indicates a deleted reference

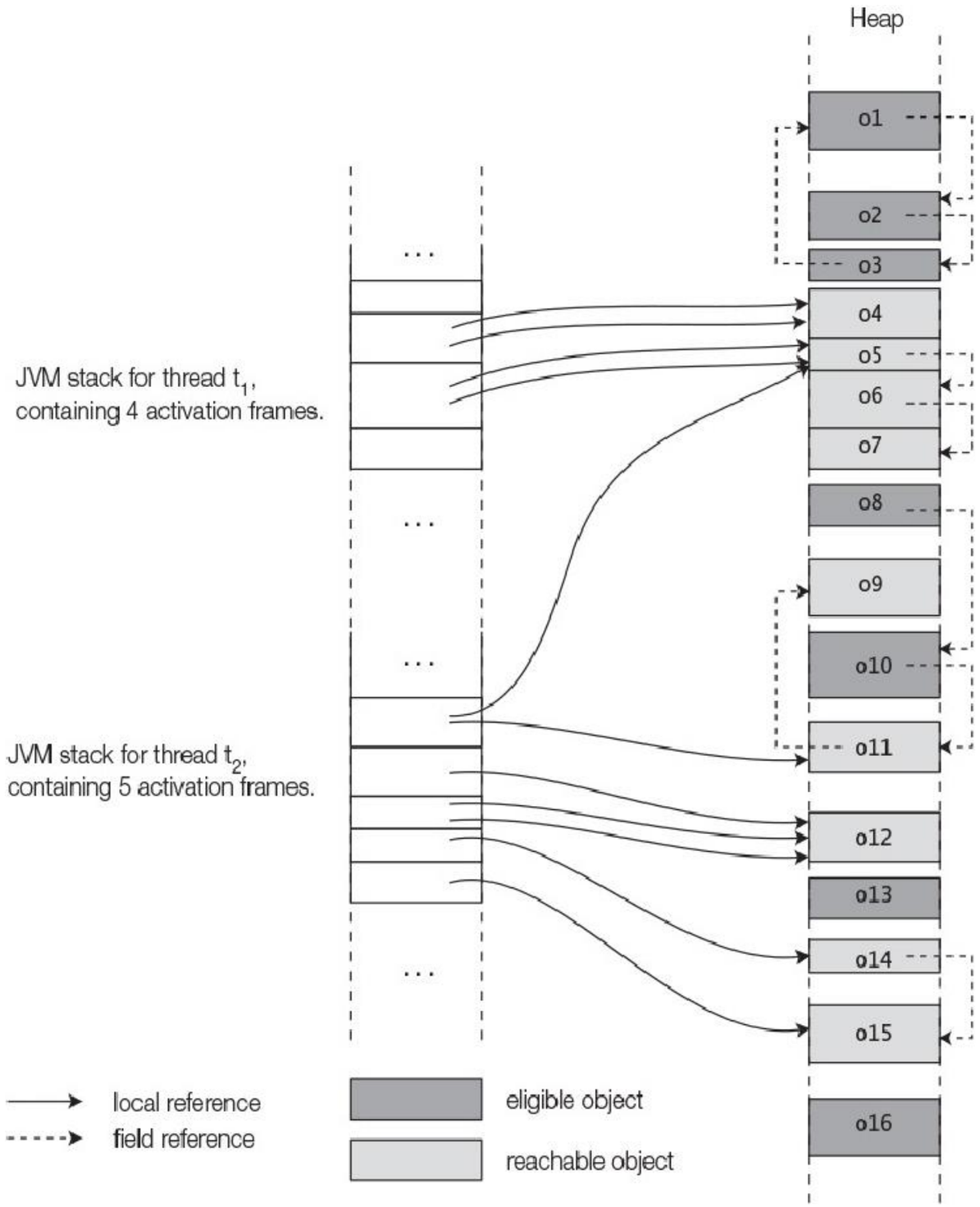
```
public class Lost {
  public static void main(String [] args) {
    Lost x = new Lost ();
    x = null;
    doComplexStuff();
  }
}
```

Стек и куча

```
 1. class Collar { }
 2.
 3. class Dog {
 4.   Collar c;                          // instance variable
 5.   String name;                       // instance variable
 6.
 7.   public static void main(String [] args) {
 8.
 9.     Dog d;                           // local variable: d
10.     d = new Dog();
11.     d.go(d);
12.   }
13.   void go(Dog dog) {                 // local variable: dog
14.     c = new Collar();
15.     dog.setName("Aiko");
16.   }
17.   void setName(String dogName) {     // local var: dogName
18.     name = dogName;
19.     // do more stuff
20.   }
21. }
```



Организация памяти



Чуть помедленнее, кони

- ✓ OutOfMemoryError возбуждается виртуальной машиной, когда она не в состоянии создавать новые объекты в «куче» (on the heap), а сборщик мусора при этом не может высвободить для JVM дополнительную память.
- ✓ Программист может запросить сборку мусора, вызвав метод System.gc() или его эквивалент Runtime.getRuntime().gc(), но дело в том, что СМ происходит в чисто автоматическом режиме, так что нет никакой гарантии, что в ходе жизненного цикла приложения СМ-процесс выполнится хотя бы раз. Всё, в чем можно быть уверенным, это в самом факте: подлежит или не подлежит тот или иной объект сборке мусора.

Эксперимент

```
1. import java.util.Date;
2. public class CheckGC {
3.   public static void main(String [] args) {
4.     Runtime rt = Runtime.getRuntime();
5.     System.out.println("Total JVM memory: "
                          + rt.totalMemory());
6.     System.out.println("Before Memory = "
                          + rt.freeMemory());
7.     Date d = null;
8.     for(int i = 0;i<10000;i++) {
9.       d = new Date();
10.      d = null;
11.    }
12.    System.out.println("After Memory = "
                           + rt.freeMemory());
13.    rt.gc();   // an alternate to System.gc()
14.    System.out.println("After GC Memory = "
                           + rt.freeMemory());
15.  }
16. }
```

Now, let's run the program and check the results:

```
Total JVM memory: 1048568
Before Memory = 703008
After Memory = 458048
After GC Memory = 818272
```

Метод finalize()

```
public class Finalizer {
  protected void finalize()  {
  System.out.println("Calling finalize");
}
public static void main(String[] args) {
  Finalizer f = new Finalizer();
} }
```

```
protected void finalize() throws Throwable
```

Один лишь раз

```
public class Finalizer {
  private static List objects = new ArrayList();
protected void finalize()  {
  objects.add(this);  // Don't do this
} }
```

Упражнение

```
1.  class Student {
2.      String name;
3.      Student(String name){
4.          this.name = name;
5.      }
6.  }
7.    public class Test {
8.        public static void main (String[] args) {
9.            Student s1 = new Student("Alice");
10.           Student s2 = new Student("Bob");
11.           Student s3 = new Student("Carol");
12.           s1 = s3;
13.           s3 = s2;
14.           s2 = null;
15.       }
16.   }
```

Which statement is true?

A. After line 14, three objects are eligible for garbage collection.
B. After line 14, two objects are eligible for garbage collection.
C. After line 14, one object is eligible for garbage collection.
D. After line 14, none of the objects is eligible for garbage collection.

Ответ: C

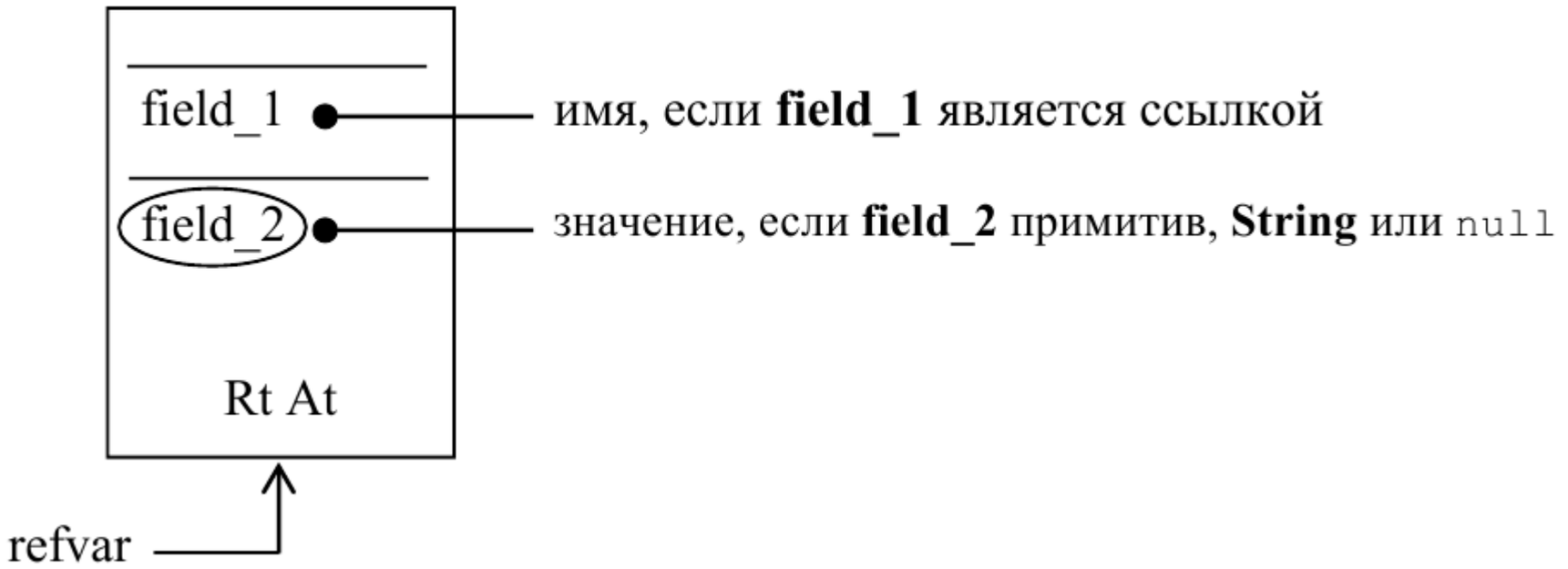
Упражнение

```
1.  class A {
2.     String name;
3.     A obj;
4.     A(String name){
5.         this.name = name;
6.     }
7.  }
8.
9.  class B extends A {
10.    B(String name) {
11.        super(name);
12.    }
13. }
14.
15. class Test {
16.    public static void main(String[] args) {
17.        A aa = new A("AA");
18.        A ab = new B("AB");
19.        B bb = new B("BB");
20.        aa.obj = ab;
21.        ab.obj = bb;
22.        ab = bb;
23.        bb = null;
24.        ab = null;
25.
26.    }
27. }
```

When the object ab, created by LOC18, becomes eligible for GC?

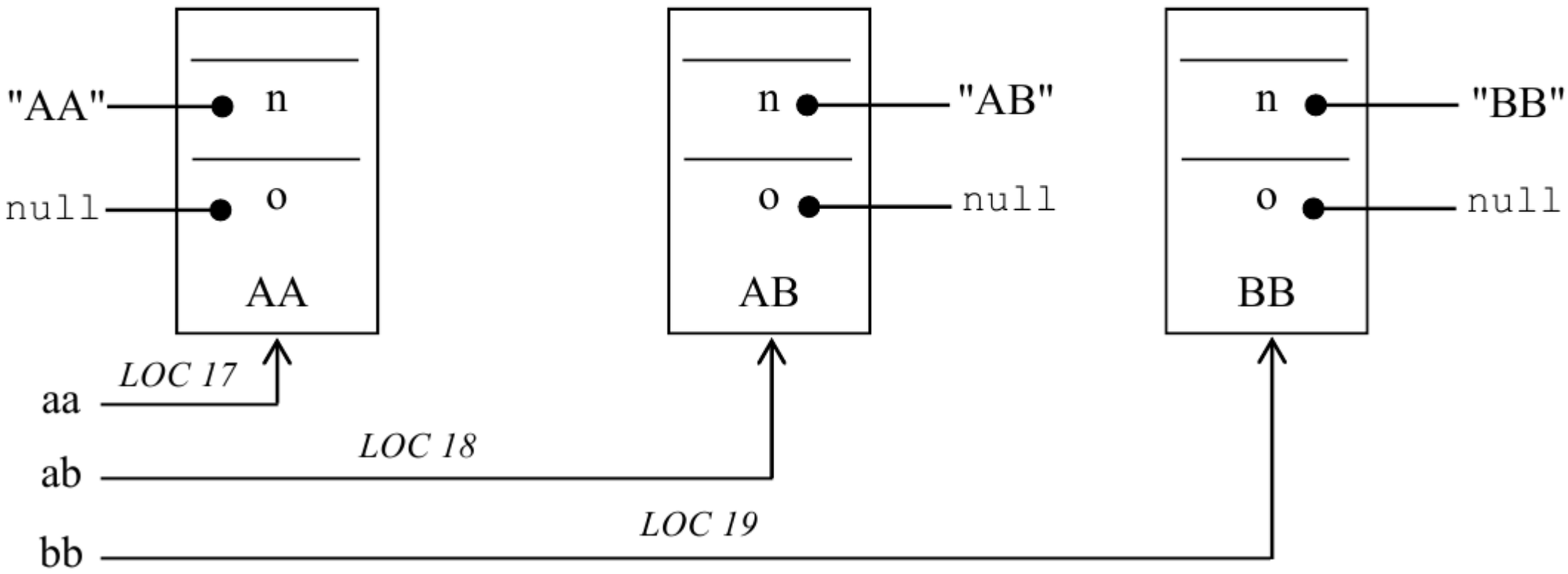
A. After line 22
B. After line 23
C. After line 24
D. Never in this program

Рисуем стрелочки



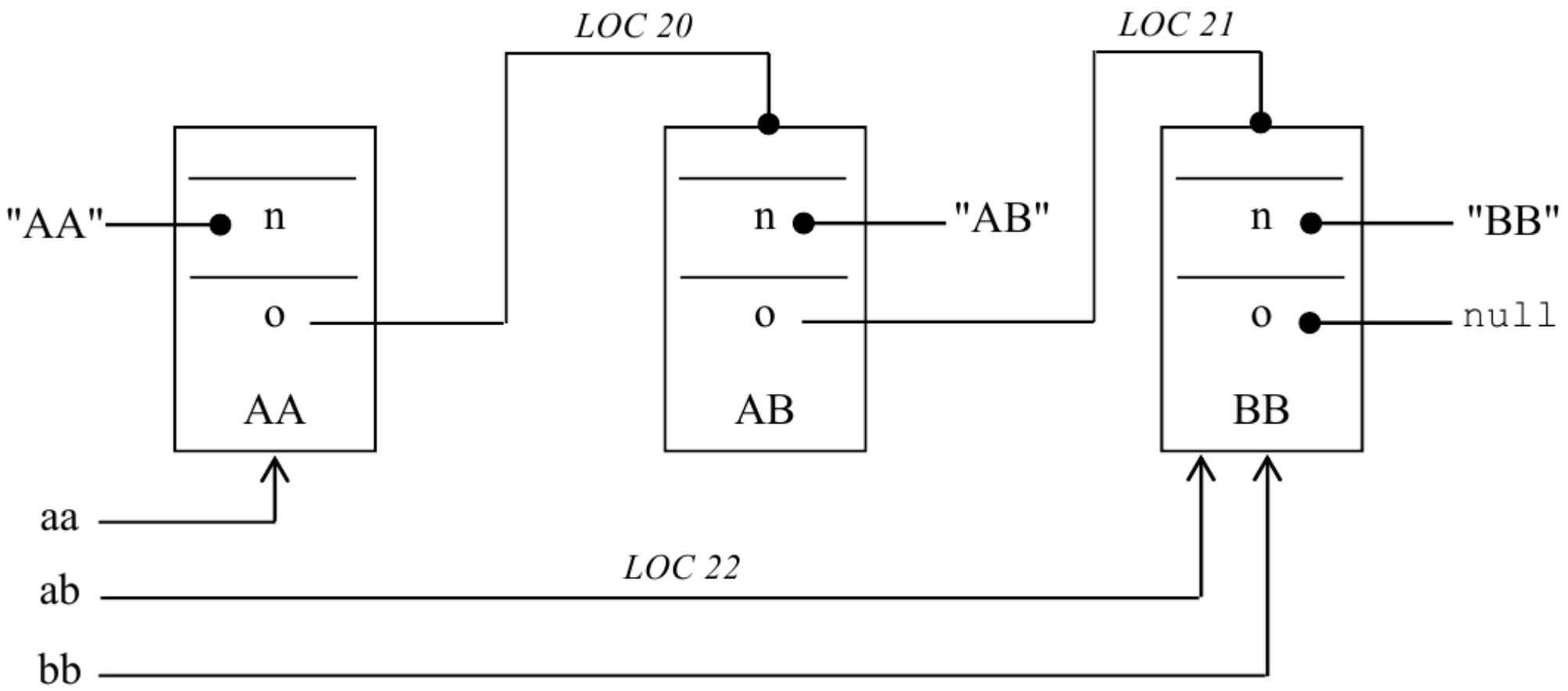
Рисуем стрелочки

Строки с 17 по 19 рисуют нам следующую картину (кстати, чтобы сэкономить время на экзамене, вместо имен полей ставятся лишь их начальные буквы, т.е. ‘n’ означает **name**, а ‘o’ означает **obj**):



Рисуем стрелочки

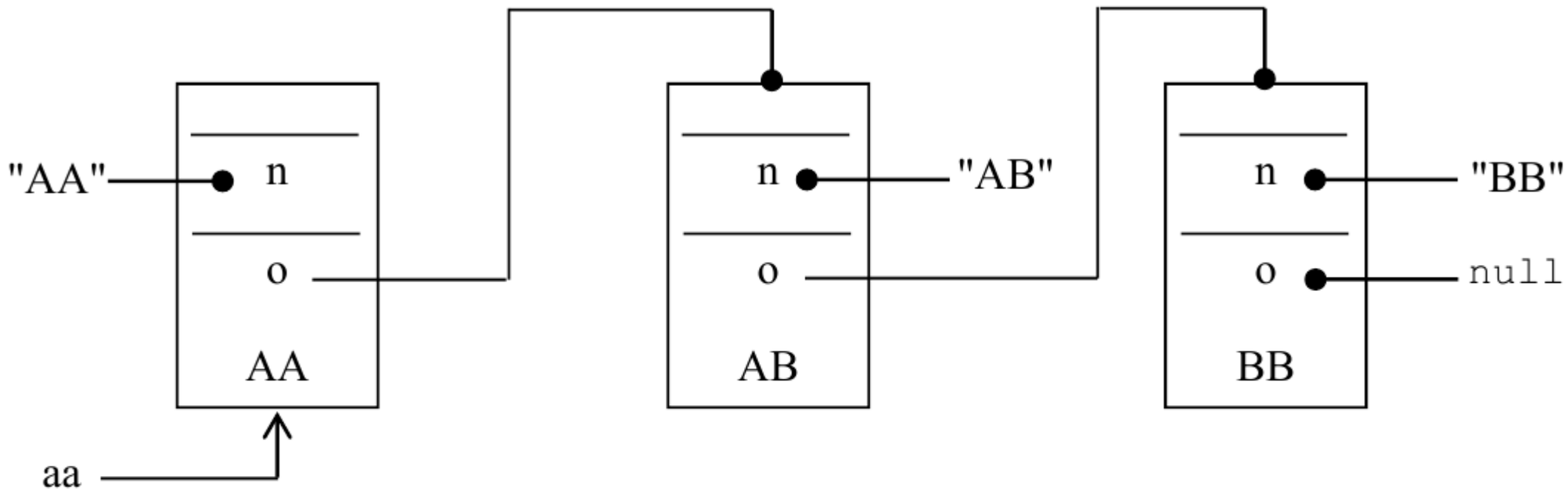
Строки с 20 по 22 приводят к вот такому состоянию:



Отметьте, что после строки 22 **ab** указывает не на **bb** (поскольку **bb** всего лишь ссылочная переменная), а на *объект*, на который ссылается **bb**.

Рисуем стрелочки

И наконец, строки 23-24 сбрасывают **ab** и **bb** в null, так что у нас остается лишь цепочка:



Эта диаграмма наглядно иллюстрирует, что все три объекта живы и прекрасно себя чувствуют вплоть до конца программы. Чтобы убедиться в этом, достаточно поставить на строке 25 операцию печати и запустить код на исполнение:

```
System.out.println("Hello from " + aa.obj.obj.name + "!");      // Hello from BB!
```

Ответ: D

Вопросы?

Домашнее задание

Тест

Домашнее задание

Жизненный цикл объекта (создание, переприсваивание ссылки и сборка мусора)

Цель: Закрепление материала вебинара с помощью прохождения теста, аналогичного экзаменационному.

1. Пройдите, пожалуйста, тест (9 заданий): https://forms.gle/jg8S8f9hp4F23Hcm7
2. Сообщите о прохождении в Чате с преподавателем.

Критерии оценки: Тест считается пройденным, если результат - выше 65%.

Рекомендуем сдать до: 30.04.2021

Статус: не сдано

Чат с преподавателем

Пожалуйста, пройдите опрос

https://otus.ru/polls/17813/

Спасибо за внимание!

Долгой жизни Вашим объектам!