Java ABC

Giorgio Brajnik

giorgio.brajnik@uniud.it

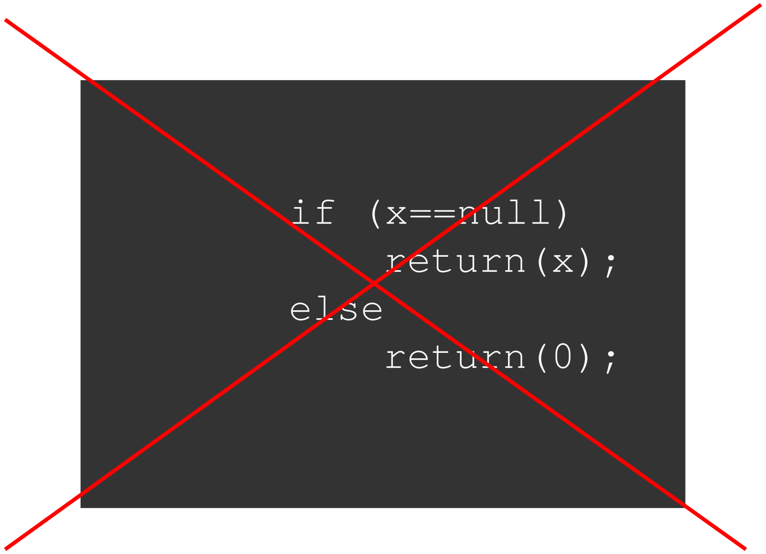
Ott 2021

Java Style Rules

Java style rules

- package names
  - **camelCase**, start with lowercase
- class names → **CamelCase**
- method and variable names
  - **camelCase**, start with lowercase
  - methods start with a verb → **enrollStudent**
- blocks of code
  - always surrounded by {}

```
if (x==null){
    return(x);
} else {
    return(0);
}
```



```
if (x==null)
    return(x);
else
    return(0);
```

On names

- use **intention revealing** names

```
int elapsedTimeInDays;

int[] source, destination;
```

~~`int d;`~~

~~`int[] a1,a2;`~~

- avoid **Disinformation**

~~`XYZControllerForEfficientHandlingOfStrings`~~

~~`XYZControllerForEfficientStorageOfStrings`~~

On names

- use **pronounceable** names

```
String generationTimeStamp;

String modificationTimeStamp;
```

~~`String genymdhms;`~~

~~`String modymdhms;`~~

On names

- use **searchable** names

```
int final realDaysPerIdealDay = 4;
int final WORK_DAYS_PER_WEEK = 5;
int NUMBER_OF_TASKS = 34;
int sum = 0;
for (int j=0; j < NUMBER_OF_TASKS; j++){
    int realTaskDays = taskEstimateInIdealDays[j] *
                        realDaysPerIdealDay;
    int realTaskWeeks = realDays/WORK_DAYS_PER_WEEK;
    sum += realTaskWeeks;
}
```

```
for(int=j;j<34;j++)
    s+= (t[j]*4)/5
```

On names

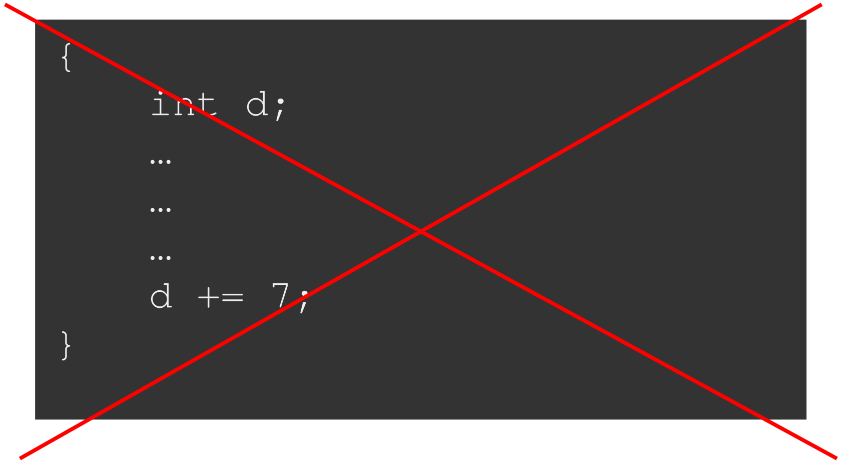
- avoid **encodings**

```
String phoneString;

public class Part {
        private String m_dsc;
        …

List<String> DataList;
```

On names

- use **meaningful** names
  - for **classes** → use concrete nouns (Customer, Account, WikiPage, AddressParser)
    - avoid using Manager, Processor, Data, Info
  - for **methods** → use verbs (save, deletePage, getName,setName, isName)
  - for **variables**:
    - length of name is proportional to scope size
    - AVOID

      O (capital o),
      I/l (capital I, lowercase L)



```
{
    int d;
    …
    …
    …
    d += 7;
}
```

Java Overview

Overview

- classes and objects
  - with fields/members
    - eg. aStudent.name
  - with methods
    - eg. aStudent.getName()
    - aStudent.setName(“Pinco”)
    - aStudent.registerExam(exm)
- class ~ category
- instance ~ thing belonging to a category

Classes

```
class MyClass extends MySuperClass implements YourInterface {
    // field, constructor, and
    // method declarations
}
```

- a class defines
  - class and instance variables
  - constructors
  - other methods
    - accessors
    - modifiers

Class vs instance variables

```
public class Student {
        // instance variables
        private String name;
        private int id;


        public void setId(int newId){
            id = newId;// or this.id = newId;
        }

        // a class variable
        static int numberOfEnrolledStudents = 0;
        …
        Student.numberOfEnrolledStudents ++;

        // a constant
        public static final String degreeName = “Laurea Informatica”;

        …
    }
```

Packages

```
…    package it.uniud.poo.utilities;
     public class Grades {
         …
         public static int[] getScale(){…}
}
```

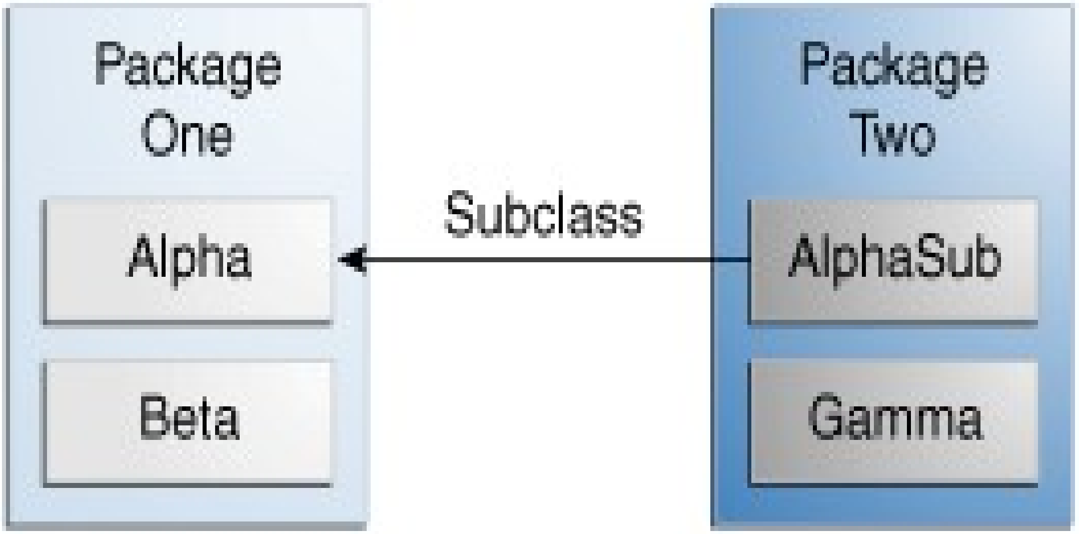
```
…    package it.uniud.poo.students;
     import it.uniud.poo.utilities.Grades;
     import it.uniud.professors.School.getProfessors;
     import it.malignani.professors.School.getProfessors;
     import it.uniud.poo.*;
     public class Student {
         …
         votiPossibili = Grades.getScale();
         …
         profsUD = it.uniud.professors.School.getProfessors(prorog1516);
         profsMalignani =
             it.malignani.professors.School.getProfessors(prorog1516);
         …
     }
```

Packages

- symbolic names
  - fully qualified or not
  - used as “containers”
    - to reduce/resolve conflicts
- visibility rules
  - <normal: package level>
  - public
  - private
  - protected

Visibility of classes

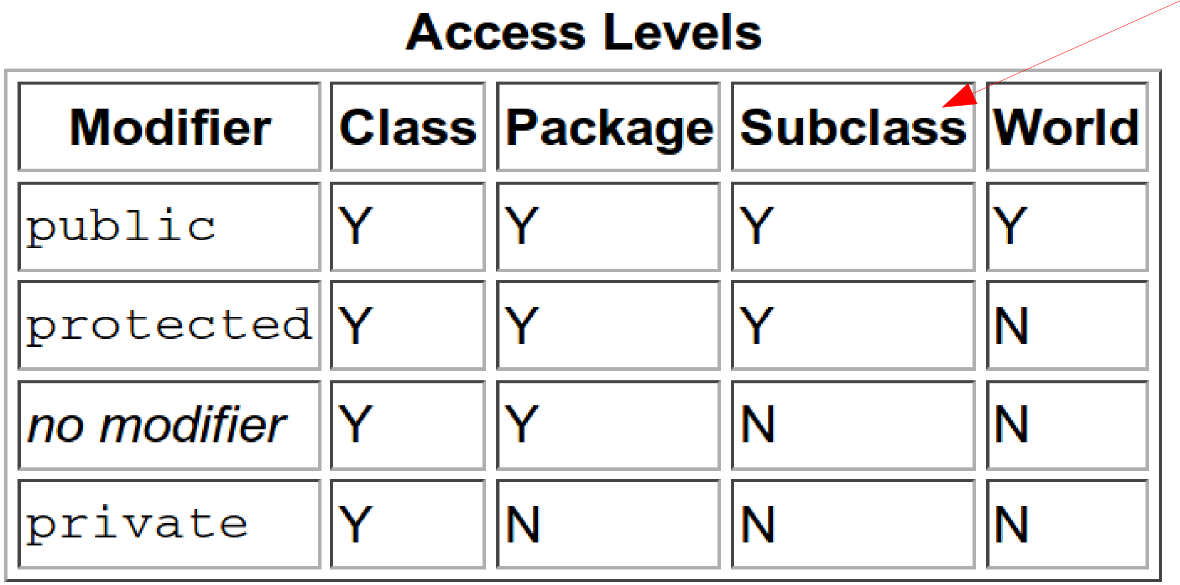
- **public**
  - can be imported in other packages
- **(no modifier)**
  - can be used only within the package where it is defined



Visibility

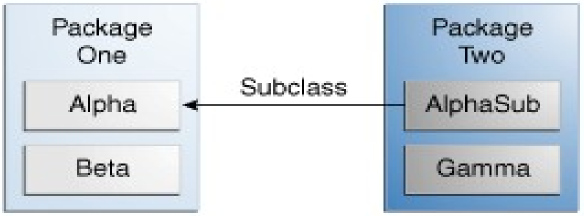
For **methods** and **fields**

Subclasses outside the package



Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N



Rules:
- whenever possible use “private”
- for fields ALWAYS use private
- use “public” only for constants meant to be exported elsewhere

(https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html)

Local variables

Stack:
activation records with

- values of variables
- return address
- returned value

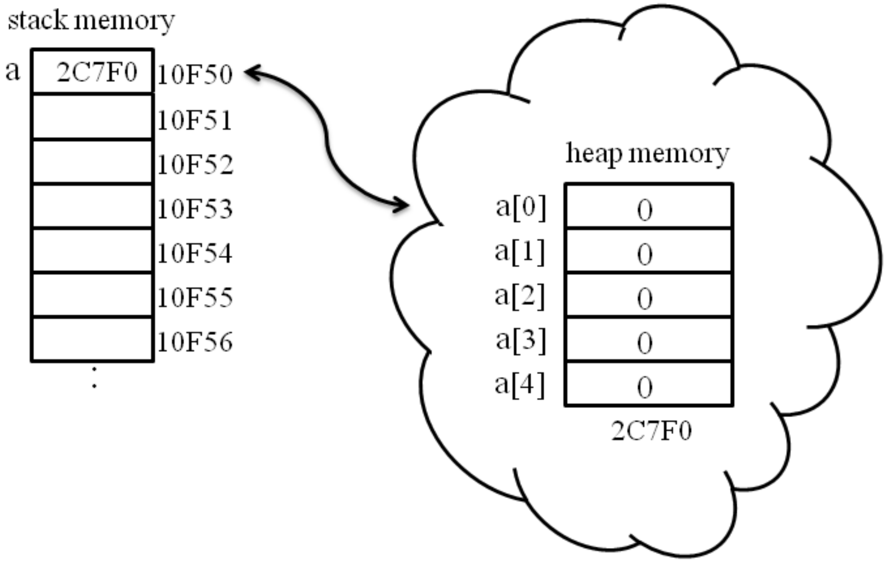
Policy: LIFO

Heap:
dynamic variables with

- arbitrary values
- their address is stored elsewhere (heap or stack)

Policy: garbage collection

```
int a[]=new int[5]; // declaring an integer array
```



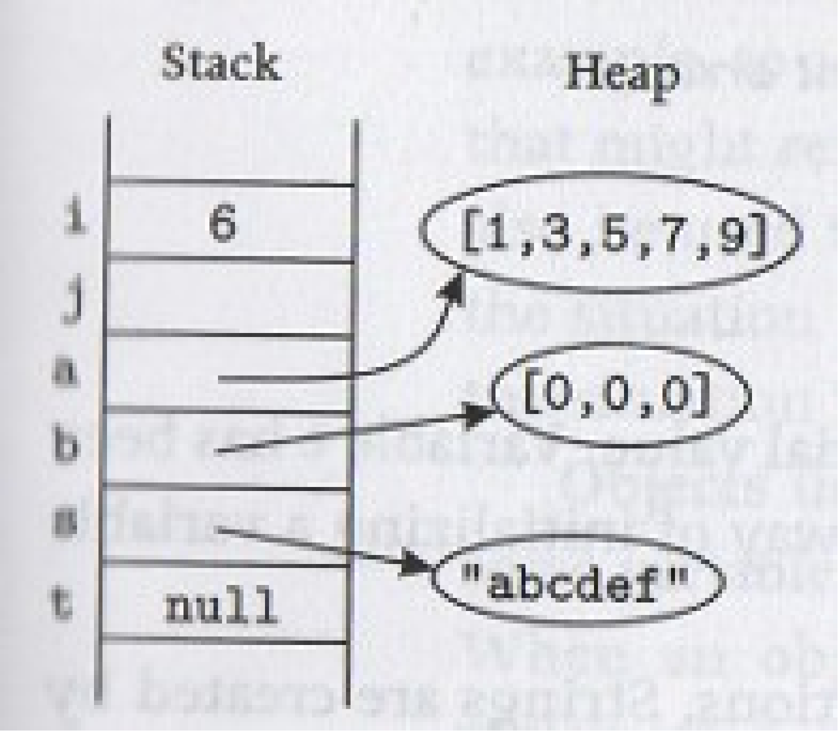
when int a []= null there will be no instances for variable a

(stackoverflow.com)

Mutability

```
int i,i2 = 0;
int j;
int [] a,b;
String t;

…

i = 6;
System.out.println(j);
b = new int[3];
a = {1,3,5,7,9};
String s = “abcdef”;

j = i;
b = a;
t = s;
```

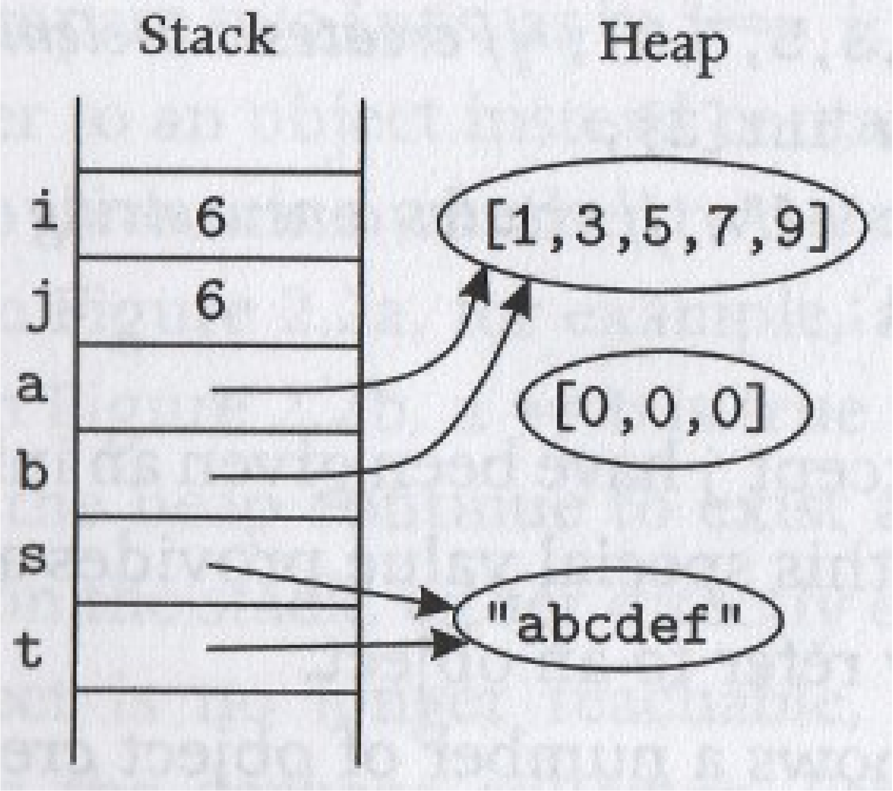
What happens in memory **before** the last 3 statements are executed?



What happens in memory when the last 3 statements are executed?

Mutability

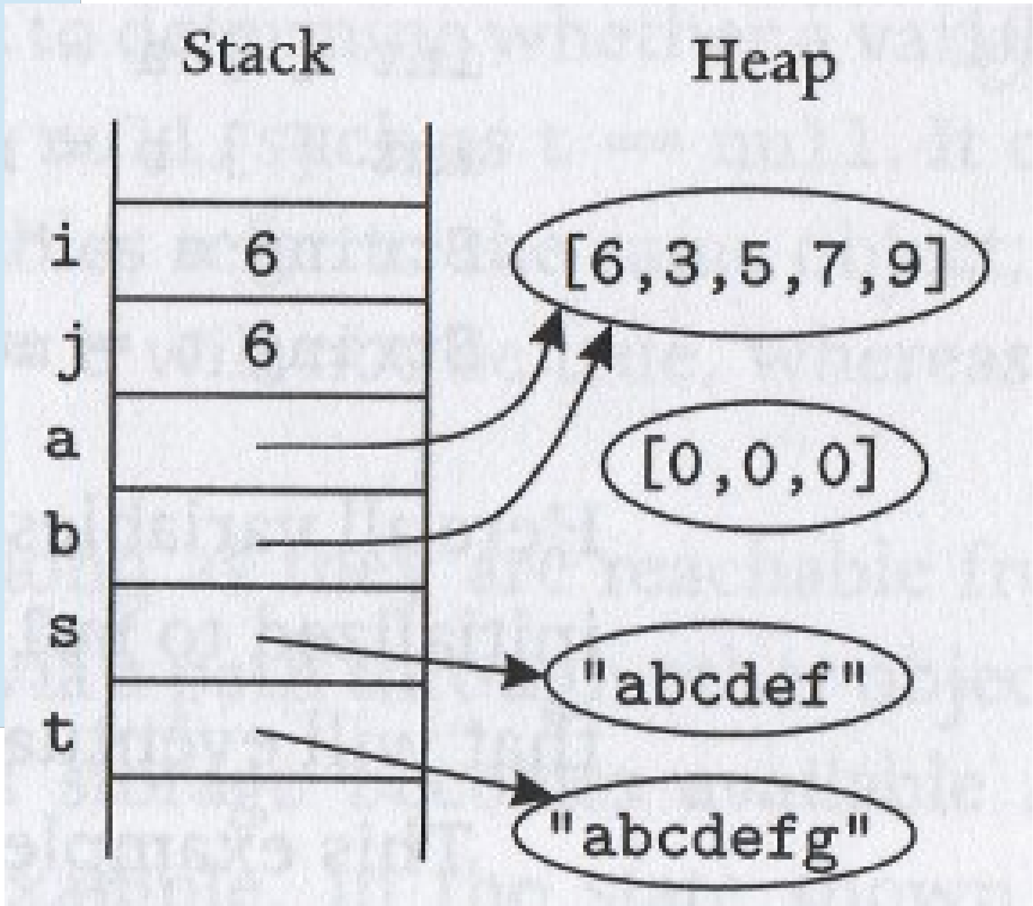
```
int i,i2 = 0;
int j;
int [] a,b;
String t;

…

i = 6;
System.out.println(j);
b = new int[3];
a = {1,3,5,7,9};
String s = “abcdef”;

j = i;
b = a;
t = s;

t = t + “g”;
```



What happens in memory when the last statement is executed?

String is immutable

```
int i,i2 = 0;
int j;
int [] a,b;
String t;

…

i = 6;
System.out.println(j);
b = new int[3];
a = {1,3,5,7,9};
String s = “abcdef”;

j = i;
b = a;
t = s;

t = t + “g”;
```



Mutability

- Objects are either mutable or immutable
- **Mutable**
  - their state can change over time
  - eg array, Student, ...
- **Immutable**
  - their state never changes
  - eg String, …
- **Shared** object:
  - its reference is stored in 2+ variables

Identity and equality

i == j → true

a == b → true

a.equals(b) → true

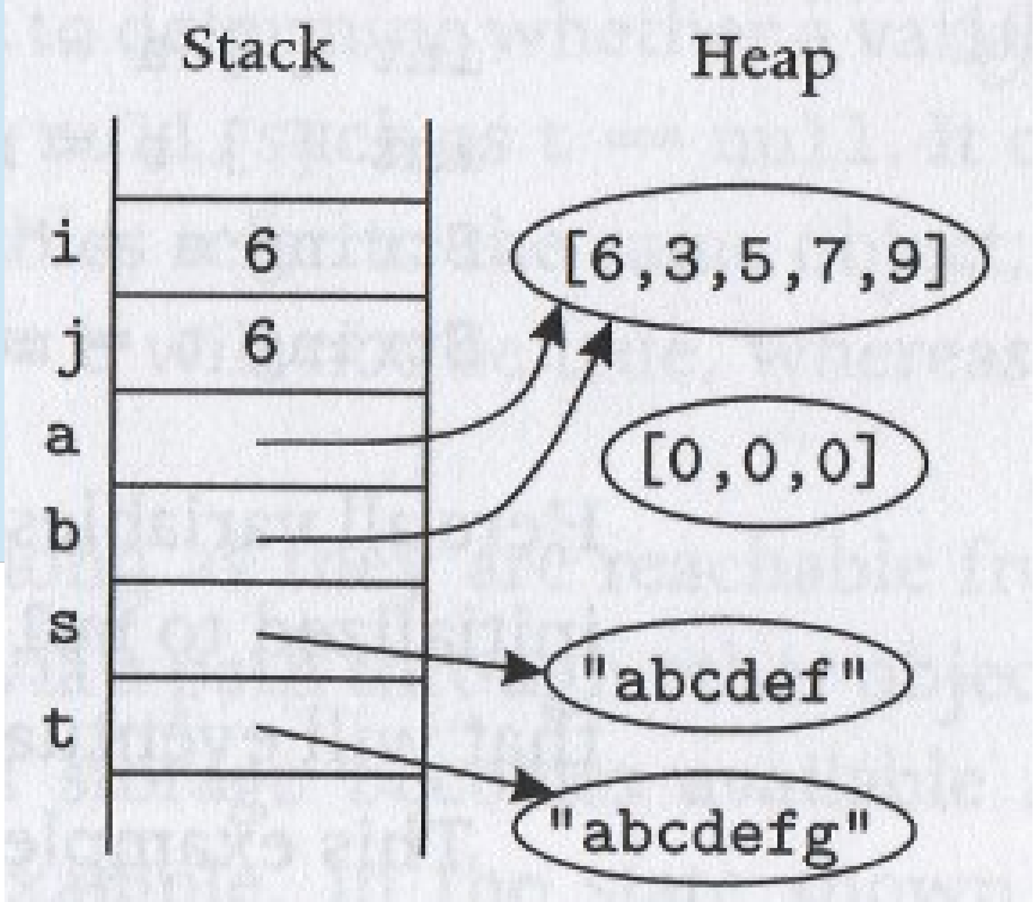
i.equals(j) → ERROR

(s + “g”) == t → false

(s + “g”).equals(t) → true

t = s.clone()

t.hashCode()



Objects

```
…    package it.uniud.poo.students;

          …

          aStudent = new Student();
          aStudent.name = “Pinca”;
          aStudent.setName(“Pinca”);
          List<Exam> studExams = aStudent.getExams();
          anotherStudent = esse3.retrieveStudent(“12345”);
          studExams.get(0).setRating(ItalianUniversity.trentaELode);
          if (aStudent == anotherStudent){ …}
          if (aStudent.equals(anotherStudent)) { … }
     }
```

Shallow and deep equality

- What happens with an object that contains references to other objects?

```
class Student {
      public String name;
      public String fiscalCode;
}

…

class Course  {
      private String name;
      private List<Student> enrolledStudents = new ArrayList<Student>();
      …
      public enrollStudent(Student studentToEnroll) { … }
}
```

Shallow and deep equality

```
// in some method
…
        Student s = new Student(“John Smith”,”sthjhn434E444922R43”);
        Course c = new Course(“POO”);
        c.enrollStudent(s);
        …
}
```

- Course is mutable (it has a modifier)
- If it had no modifiers it would be immutable
- PROVIDED its students are immutable too
  - which is not the case

```
        …
        List<Student> enrolled = c.getEnrolledStudents()
        Student s = enrolled.get(0);
        s.name = “Frank Smith”; // we changed state of s and of c
```

Method calls

```
…   package it.uniud.poo.students;

    studExams.get(0).setRating(mean(anotherStudent.getAllRatings());


EXPRESSION.METHOD(EXPR1, EXPR2, ...)
```

- Call by value
  - evaluate expression, then expr1, then expr2
  - call method “METHOD” of object that is the value of EXPRESSION and that has an appropriate signature
- and if there's no object that is the value of EXPRESSION?
  - → NullPointerException

Method calls

```
private static int swap(int[] a, int k)
…
x = swap(arr1, j);
return(x);
```

- Formal parameters
  - they work as if they were local variables
- Actual parameters
  - as if it the call were an assignment
    - a ← arr1; k ← j;
    - (returned value) ← x (at the end)