# IN1007 Programing in Java

## Lecture 5 (Week 8): The Life Cycle of an Object

## Announcements

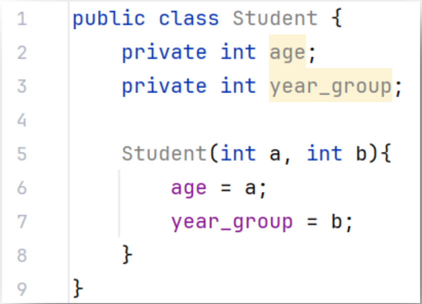
- **Second coursework available NOW** → deadline 4th Dec at 5pm (see details on Moodle)

- Viva details

## Today’s Lecture

- Object construction (in more detail)
  - Instance fields
  - Static fields
- Object destruction
- Some Quiz questions

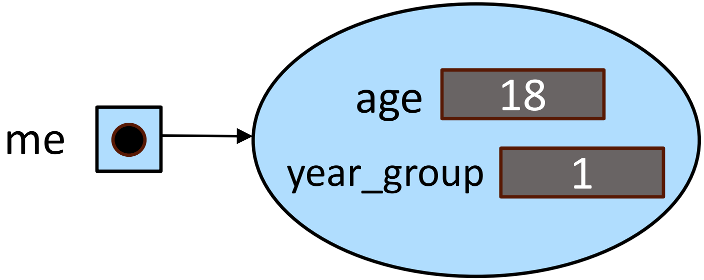
## Object Construction



```java
public class Student {
    private int age;
    private int year_group;

    Student(int a, int b){
        age = a;
        year_group = b;
    }
}
```

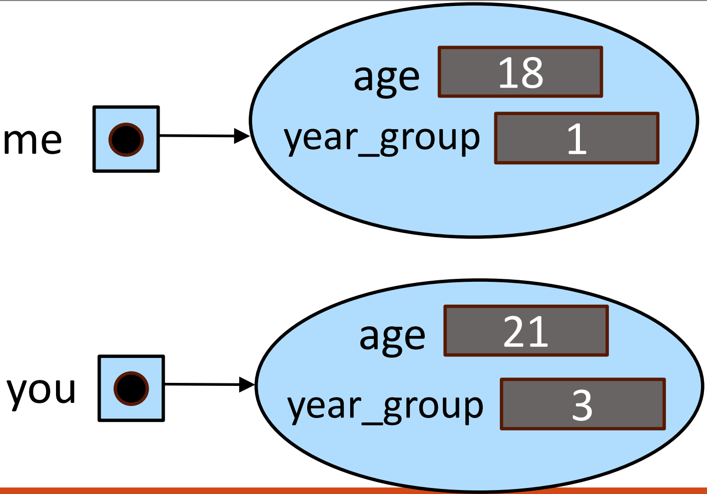
What happens when you call:

```java
Student me = new Student(18,1);
Student me = new Student(18,1);
```

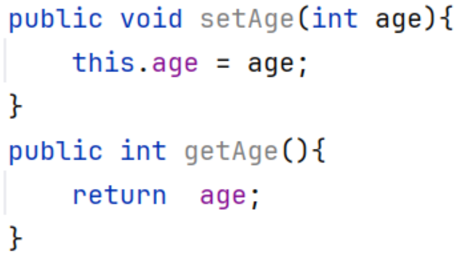


What happens when you then call:

```java
Student you = new Student(21, 3);
Student me = new Student(18,1);
Student you = new Student(21, 3);
```

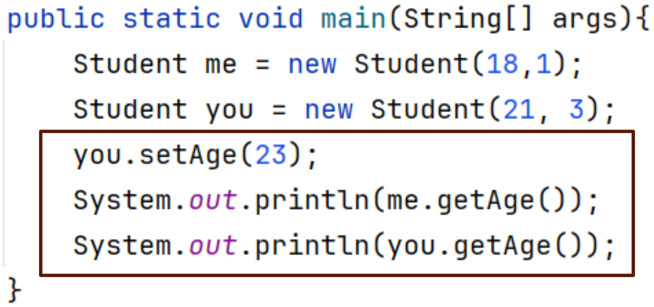


## Let’s add a getter and a setter:



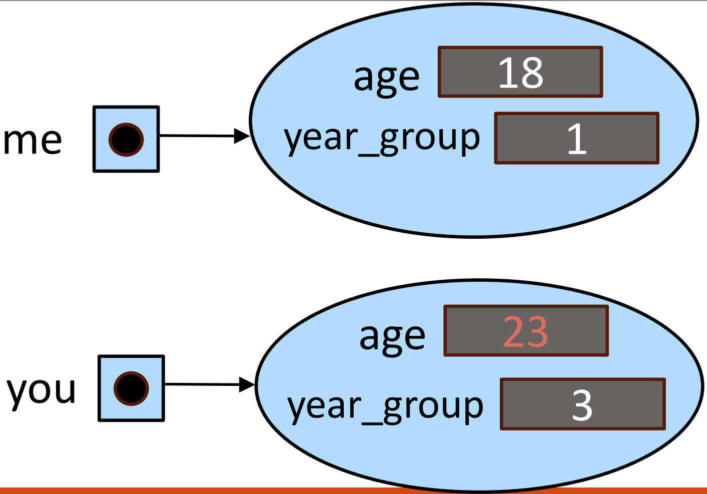
```java
public void setAge(int age){
    this.age = age;
}
public int getAge(){
    return  age;
}
```

## Now in your main() method add:

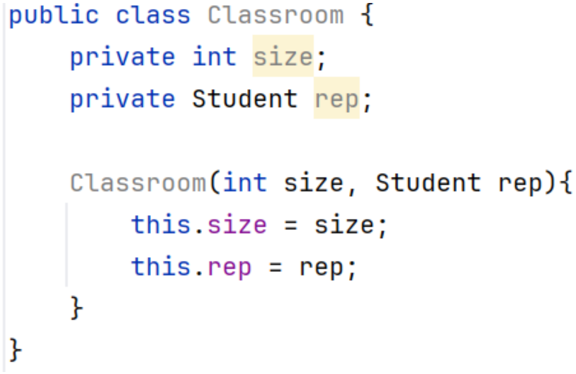


```java
public static void main(String[] args){
    Student me = new Student(18,1);
    Student you = new Student(21, 3);
    you.setAge(23);
    System.out.println(me.getAge());
    System.out.println(you.getAge());
}
```

## Observe…

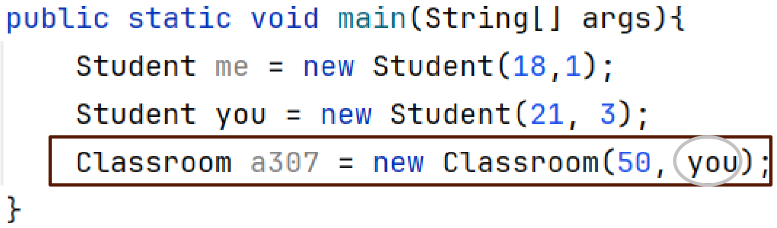


## Let us create another class



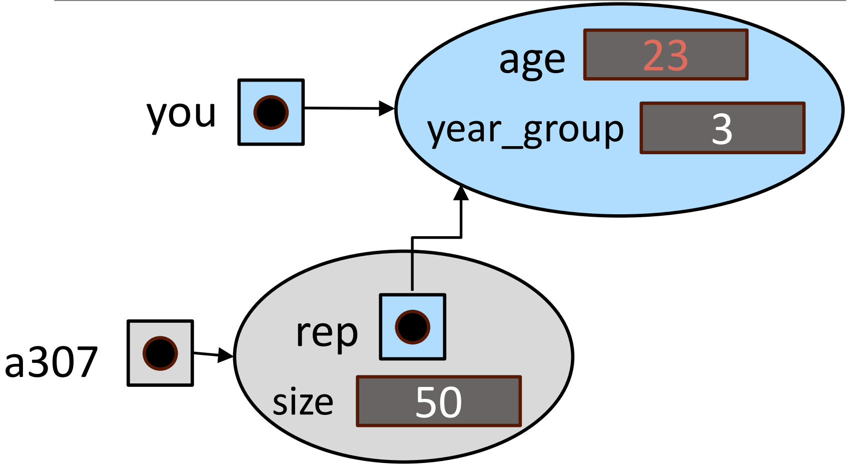
```
public class Classroom {
    private int size;
    private Student rep;

    Classroom(int size, Student rep){
        this.size = size;
        this.rep = rep;
    }
}
```

## What happens when you call:



```
public static void main(String[] args){
    Student me = new Student(18,1);
    Student you = new Student(21, 3);
    Classroom a307 = new Classroom(50, you);
}
```

## Observe…



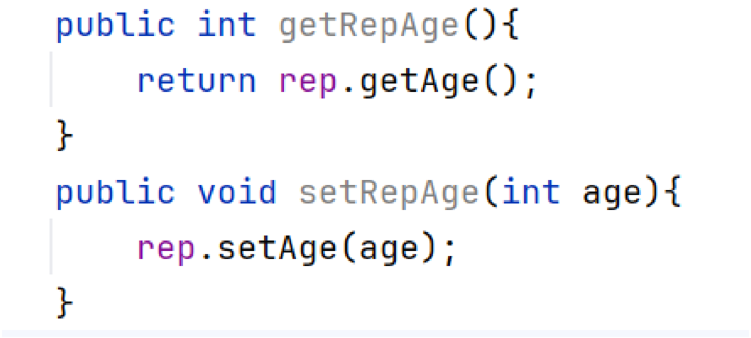
## Exercise:

See if you can add two methods to the Classroom class:

`getRepAge()` returns the age of the class rep

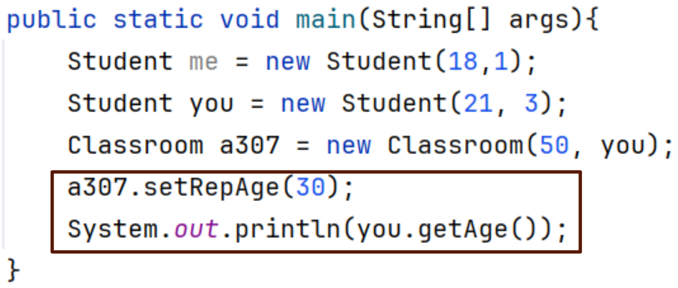
`setRepAge()` changes the age of the class rep

## Solution (in Classroom class):



```
public int getRepAge(){
    return rep.getAge();
}
public void setRepAge(int age){
    rep.setAge(age);
}
```

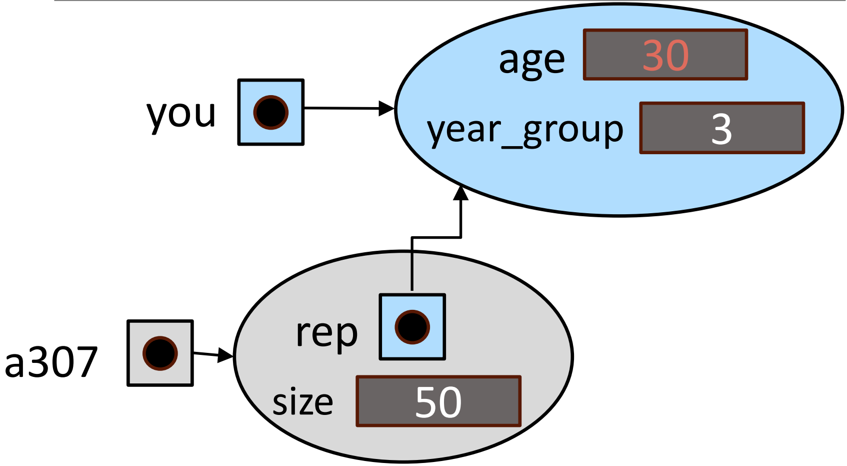
## What will happen if you run…



```java
public static void main(String[] args){
    Student me = new Student(18,1);
    Student you = new Student(21, 3);
    Classroom a307 = new Classroom(50, you);
    a307.setRepAge(30);
    System.out.println(you.getAge());
}
```

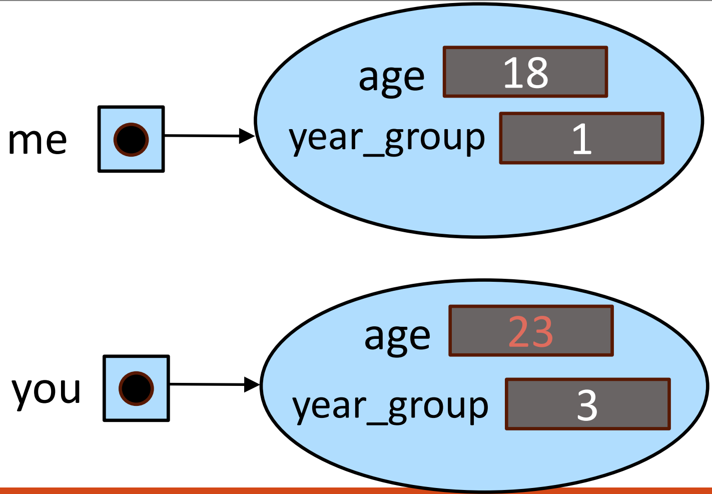
## Observe...

Two references pointing to **the same** object



## Contrast…

Two references pointing to **different** objects



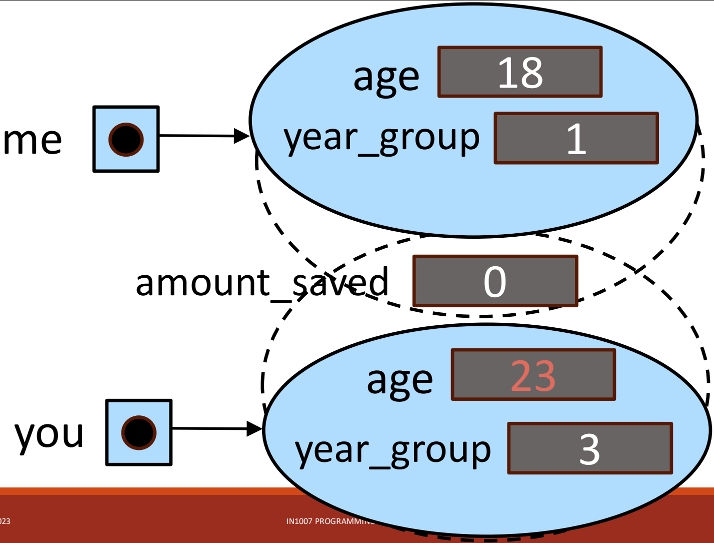
## Contrast…

Two references pointing to **different** objects

**<u>Question:</u>** What if we want two or more objects of the same type to share information?

**Example:** if we want students to raise money for an event and keep track of how much more is needed. The amount saved so far should be visible by all instances of the class `Student`

```
public class Student {
    private int age;
    private int year_group;
    static int amount_saved =0;
```



## `Static` variables and methods in Java

- Often referred to as **class-level** variables
- When a **variable** is declared as **`static`**, there is only one copy of it for the entire class (for all its instances).
- When a **method** is declared as **`static`**, it can be called without creating an instance of the class.

- Advantages:
  - Memory efficiency
  - Global access
  - Object independence,…

## Referencing variables

- Instance variables / fields are referenced by

  **`ObjectName.variableName`**

  **Example:** `me.year_group`

- Static variables / class variables are referenced by

  **`ClassName.variableName`**

  **Example:** `Student.amount_saved`

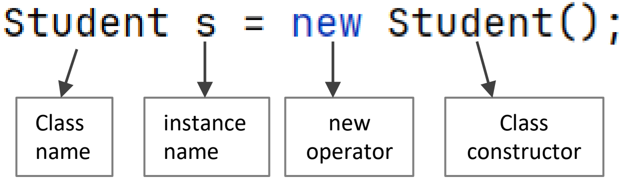
## Today’s Lecture

- Object construction (in more detail)
  - Instance fields
  - Static fields
- **Object destruction**
- Some quiz questions

## The Java Object Lifecycle

## An Object is Born!

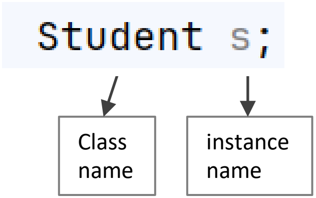
At what point in this statement do you think the object will begin to exist in memory?



```java
Student s = new Student();
```

## What if…

We only declare an object without creating a new instance?

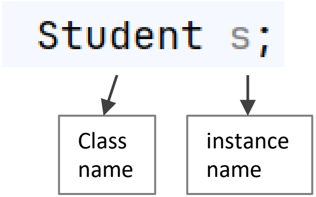


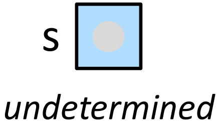
```
Student s;
```

# **Declaring** versus **Initializing** an object

We only declare an object without creating a new instance?

```
Student s;
```

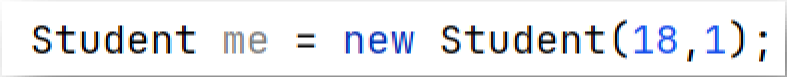




```
java: variable s might not have been initialized
```

## Initializing an Object

A call to a valid constructor must be in place:



```
Student me = new Student(18,1);
```

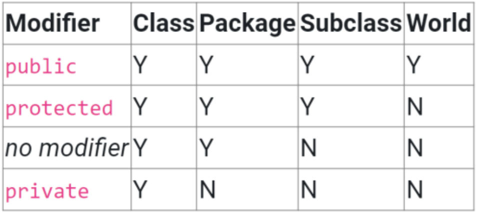
Initialization of member variables should happen inside the constructor

## Exercise (A)

Add a new constructor for the class Student to accept a name for each student

## While an Object is Alive…

- It has not gone out of scope yet
- You can access its member fields using objectName.fieldName
- Remember the principle of encapsulation



| 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 |

## Exercise (B)

Change the access modifier for the `Student` name field and observe its visibility in other classes in your package: `Main` and `Classroom`

## Be careful when initializing…

```
String myName = new Student("Alice",481523).name;
```

$\longrightarrow$ create an object and put its name value in the variable myName. After the instruction, you have no longer access to the object's reference.

Try removing the declaration altogether and type this:

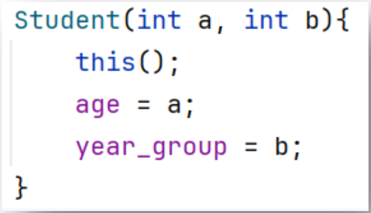
```
new Student("Alice",481523).name;
```

## While the Object is Alive…

We can use the keyword this

```
this.fieldName
this.methodName();
```

What do you reckon the call `this();` would do?



```
Student(int a, int b){
    this();
    age = a;
    year_group = b;
}
```

## Constructor Overloading

A constructor can call another using the keyword `this()`

**Exercise (c):** try adding a constructor for Student that takes only student name as parameter. It will call the default constructor which will set the age to 18 and year_group to 1 by default.

## When an Object is no longer alive...

Garbage collection: Destruction of objects that are no longer needed, ie when no longer referenced.

$\longrightarrow$ Automatic process

References that are held in a variable are usually dropped when the variable goes out of scope. Or, when an object reference is set to the special value null.

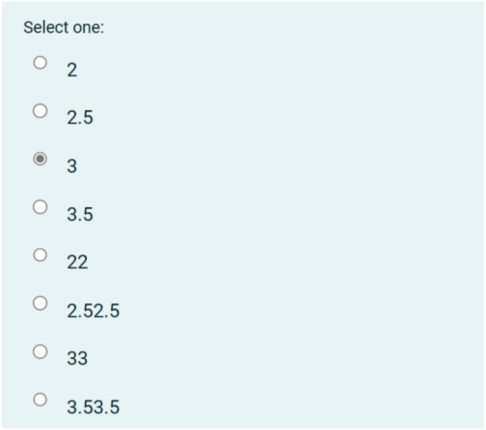
## Today’s Lecture

- Object construction (in more detail)
  - Instance fields
  - Static fields
- Object destruction
- **Some quiz questions**

Starting from a variable x with value 5 and a variable y with value 2, what would be displayed by a call to foo(x,y) followed by a call to bar(x,y) in the main method?

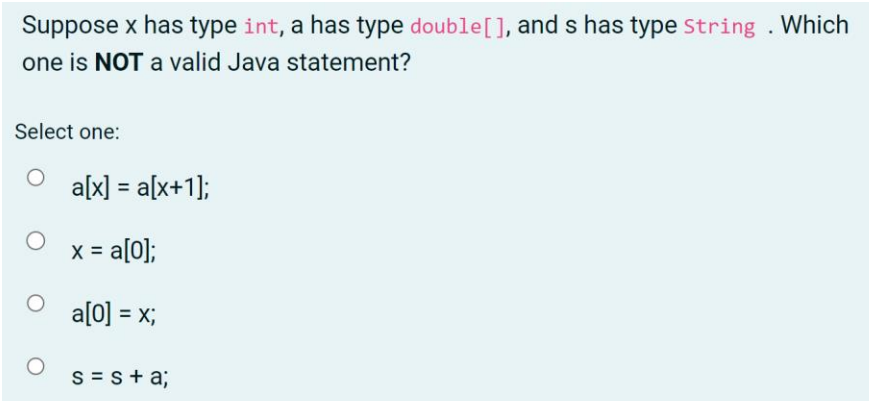
```
public static int foo(int x, int y){
    int z = x/y;
    x=y;
    z = z+1;
    return z;
}
public static void bar(int x, int y){
    System.out.println(foo(x,y));
}
```

## Correct Answer



Select one:

- ○ 2
- ○ 2.5
- [✓ 3]
- ○ 3.5
- ○ 22
- ○ 2.52.5
- ○ 33
- ○ 3.53.5



Suppose x has type `int`, a has type `double[]`, and s has type `String` . Which one is **NOT** a valid Java statement?

Select one:

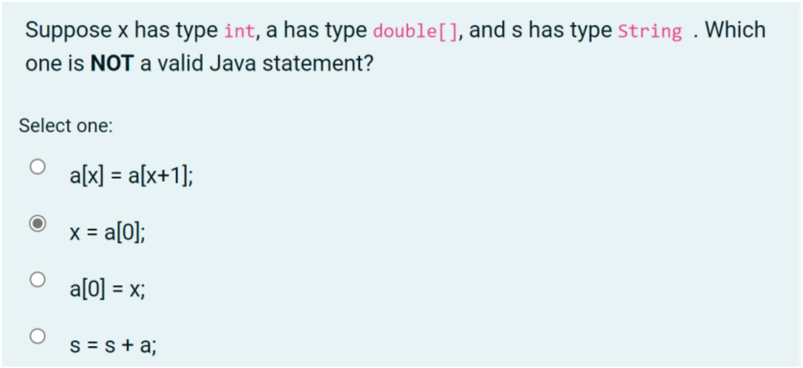
○ a[x] = a[x+1];

○ x = a[0];

○ a[0] = x;

○ s = s + a;

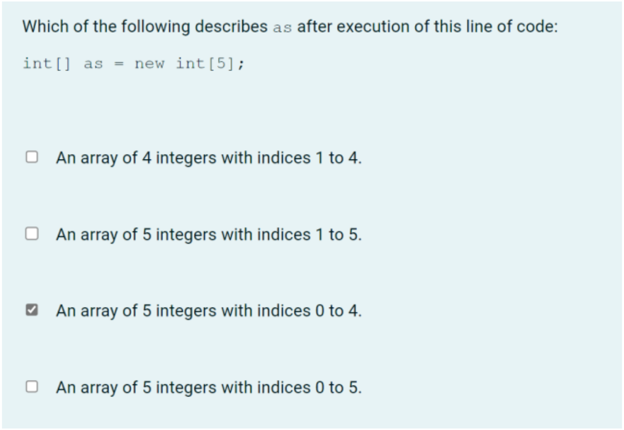
## Correct Answer



Suppose x has type `int`, a has type `double[]`, and s has type `String` . Which one is **NOT** a valid Java statement?

Select one:

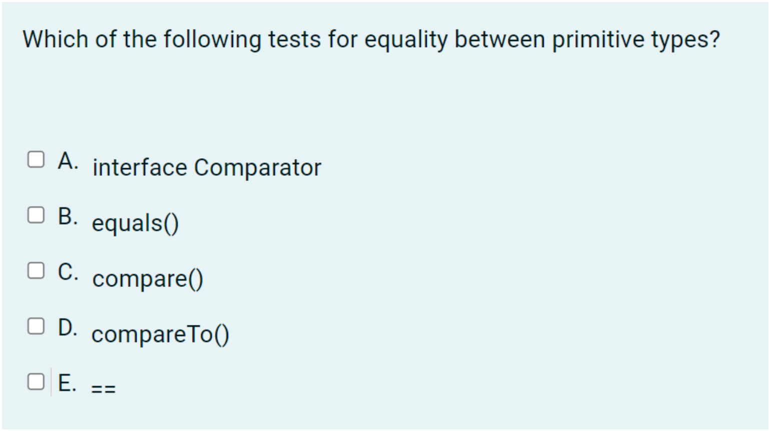
- ○ a[x] = a[x+1];
- [✓ x = a[0];]
- ○ a[0] = x;
- ○ s = s + a;



Which of the following describes `as` after execution of this line of code:

```
int[] as = new int[5];
```

- An array of 4 integers with indices 1 to 4.
- An array of 5 integers with indices 1 to 5.
- [✓ An array of 5 integers with indices 0 to 4.]
- An array of 5 integers with indices 0 to 5.



Which of the following tests for equality between primitive types?

☐ A. interface Comparator

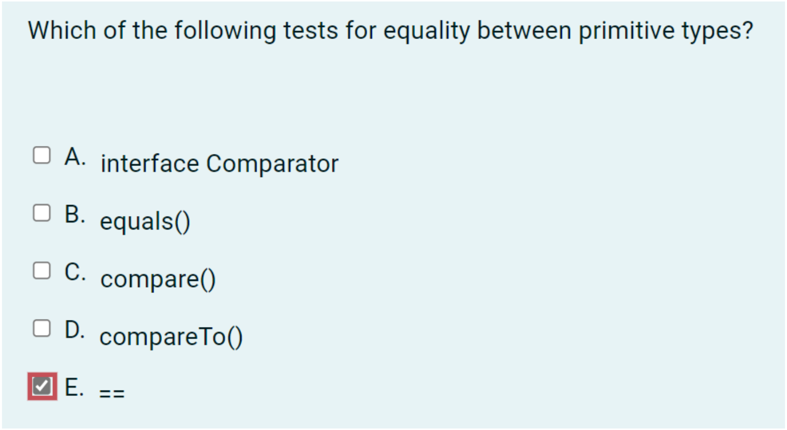
☐ B. equals()

☐ C. compare()

☐ D. compareTo()

☐ E. ==

## Correct answer



Which of the following tests for equality between primitive types?

- A. interface Comparator
- B. equals()
- C. compare()
- D. compareTo()
- [✓ E. ==]

## Exercise

1. Implement a public class Restaurant with the following private fields:
   - name of type String,
   - numberOfTables of type int.
2. Provide a constructor with one parameter of type String and that initializes the field numberOfTables to 10.
3. Provide getter and setter for the fields.
4. Modify the setter method for numberOfTables so that this field is updated only if the argument is a positive integer.
5. Add an initialization block that prints "Welcome to the new restaurant".
6. Implement another public class Test with a main method in which you create a new object of the class Restaurant and test your setter and getter methods.

## Today’s Lecture

- Object construction (in more detail)
  - Instance fields
  - Static fields
- Object destruction
- Some Quiz questions