Object Oriented Programing 1

Lecture 8: Arrays and Ways to create objects in Java

By

Elubu Joseph

MSci.IS

Email: josebulinda@gmail.com

or

jose@kumiuniversity.ac.ug

Agenda

1. Array in Java and
2. Ways to create object in Java

Introduction to Arrays in Java

An Array is a collection/list of elements of similar data types. Array is a container of object that hold values of **homogeneous** type. It is also known as static data structure because size of an Array must be specified at the time of its declaration.

Array starts from **zero** index and goes to **n-1** where **n** is **length** of the Array.

In Java, Array is treated as an object and **stores into heap memory**. It allows the storage of primitive values or reference values.

Array can be single dimensional or multidimensional in Java.

Features of Array

1. It is always indexed. Index begins from 0.
2. It is a collection/list of elements of similar data types.
3. It occupies a continuous memory location.
4. It allows access of elements randomly.

Types of Arrays

1. Single Dimensional Arrays
2. Multi-Dimensional Arrays
3. jagged Arrays

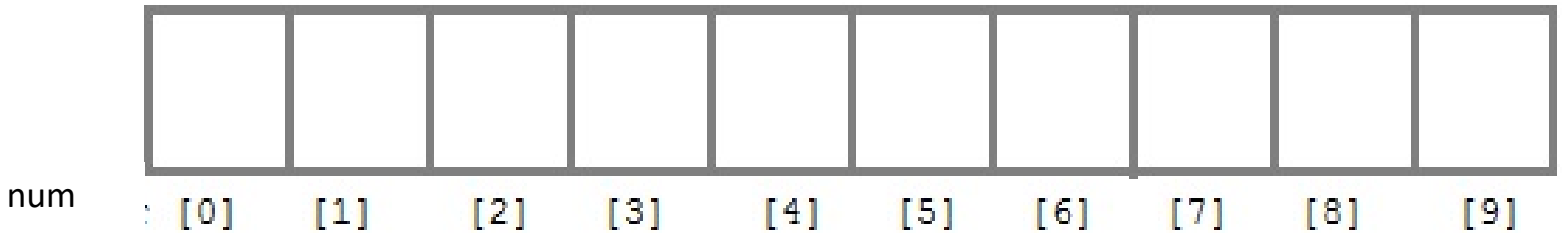
0	1	2	3

0,0	0,1	0,2
1,0	1,1	1,2
2.0	2,1	2,2

0,0	01	
1,0	1,1	1,2
2,0	2,1	2,1

Single Dimensional Arrays

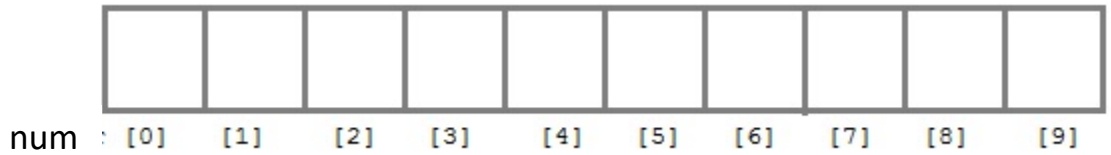
Single dimensional Arrays are Arrays that use single index to store elements. You can get all the elements of Array by just increment of its index by one.



Array Declaration

Like any other variable, arrays must be declared before they are used. Array declaration is giving of the name and definition of the Array size. Java allows declaration of Arrays by using both declaration syntax below. General form of array declaration is,

```
data-type variable-name[size];
/* Example of array declaration */
int num[10];
```



Here int is the data type, num is the name of the array and 10 is the size of array. It means array num can only contain 10 elements of int type.

Index of an array starts from **0** to **size-1** i.e first element of num array will be stored at num[0] address and the last element will occupy num[9].

Array Declaration +

The **ArrayName** can be any valid identifier and datatype can be anything like: **int, float, byte** etc depending on the type of data you want to store in a given array.

Example.

```
int ArrayName [ ];
char ArrayName [ ];
short ArrayName [ ][ ]; //2 dimensional Array
long ArrayName [ ];
int ArrayName [4];
```

Initialization of Array

Initialization is a process of allocating memory and or specifying elements to be stored by an Array, or both. At the time of initialization, we specify the size of Array to reserve memory area.

Syntax

```
ArrayName = new datatype[size];
```

Or

```
ArrayName[20]={ , , , , , , , , , , };
```

new keyword/operator is used to initialize an Array.

The **ArrayName** is the name of Array, new is a keyword used to allocate memory and size is length of Array. We can combine both declaration and initialization in a single statement.

Create An Array

Lets create a single dimensional Array.

```
class Arrayer {
public static void Arrayers() {
      int num[] = new int[10];
      for(int x : num) {
      System.out.print(x);
      }
      System.out.println();
    }
}
```

OUTPUT

```
0000000000
```

Explanation

In the above example, we created an Array **num** of **int type** and can store **10** elements. We iterate the Array to access its elements and it prints zero Ten times to the console.

It prints zero because we did not set values to Array, so all the elements of the **Array initialized to 0 by default.**

Set Array Elements

We can set Array elements either at the time of initialization or by assigning direct to its index.

```
int[] num = {10,20,30,40,50};
```

Here, we are assigning values at the time of Array creation. It is useful when we want to store static data into the Array.

Or

```
num[1] = 20;
```

Here, we are assigning a value to Array's 1 index. It is useful when we want to store dynamic data into the Array.

Sample program

Here, we are assigning values to Array by using both the ways discussed above.

```
public static void ArrayElements(){
      int[] num = {20,40,30,40,50};
      for(int x : num) {
            System.out.print(x);
      }
      // assigning a value
      num[1] = 200;
      System.out.println("\nThe Element at first index: " +num[1]);
}
```

OUTPUT

```
2040304050
The Element at first index: 200
```

Accessing Array elements

we can access Array elements by its index value. Either by using loop or direct index value. We can use loop like: for, for-each or while to traverse the Array elements.

```
public static void accessArrayElements(){
      int num[] = {20,40,30,40,50};
      for(int x=0; x<num.length; x++) {
            System.out.print(num[x]);
      }
            System.out.println("\nThe Element at last index: " +num[4]);
}
```

OUTPUT

```
2040304050
The Element at first index: 50
```

Accessing Array elements+

Here, we are traversing Array elements using loop and accessed an element randomly.

Note:-To find the length of an Array, we can use length property of Array object like: num.length.

Multi-Dimensional Array

A multi-dimensional Array is very much similar to a single dimensional Array. It can have multiple rows and multiple columns unlike single dimensional Array, which can have only one row index.

It represent data into tabular form in which data is stored into rows and columns.

Multi-Dimensional Array Declaration

```
datatype[ ][ ] ArrayName;
or
datatype ArrayName[ ][ ];
```

Initialization of Array

```
Datatype ArrayName[ ][ ] = new ArrayName[no_of_rows][no_of_columns];
```

or

```
Datatype ArrayName[ 3][ 4];
```

The **ArrayName** is the name of Array, **new** is a keyword used to allocate memory and no_of_rows and no_of_columns both are used to **set size of rows and columns** elements.

Like single dimensional Array, we can statically initialize multi-dimensional Array as well.

Initialization of Multi-Dimensional Array

```
int[ ][ ] num = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
```

This is 3*5 Array. i.e. three rows and five columns

The information stored in an Array is directed using index addresses. For example 13 in the num Array above would be placed at address 2,2.

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4

Access Multi-Dimensional Array Elements

```
public static void accessMDE() {
int num[ ][ ] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
     for(int i=0;i<3;i++) {
          for (int j = 0; j < 5; j++) {
               System.out.print(num[i][j]+" ");
          }
     System.out.println();
     } // assigning a value
System.out.println("The Element at third row and third
column: " +num[2][2]);
}
```

OUTPUT

```
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
The Element at third row and third column: 13
```

Jagged Array

Jagged Array is a type of dimensional Array that has same number of rows but different number of columns(elements). In java, a jagged Array means to have a multi-dimensional Array with uneven size of columns in it.

E.g.

```
int num[2][]={{2,3,4,5},{3,4,5,6,7,8,}};
```

The example above shows that the Array called num2[] is dimensional Array with 2 rows, with row 0 having 4 columns while row 1 having 6 columns.

Initialization of Jagged Array

Jagged Array initialization is a little different. We have to set columns size for each row independently.

Example

```
int[ ][ ] num = new int[3][ ];
num[0] = new int[3];
num[1] = new int[4];
num[2] = new int[5];
```

Sample Jagged Array program

```
public static void accessJAE() {
      int num[ ][ ] = {{39,22,32},{42,52},{62,72,82,92}};
      for(int i=0;i<3;i++) {
            for (int j = 0; j < num[i].length; j++) {
                  System.out.print(num[i][j]+" ");
            }
      System.out.println();
      }
}
```

OUTPUT

```
39 22 32
42 52
62 72 82 92
```

Objects in java

Objects in java

Java is an object-oriented language, everything revolve around the object.

An object **represents runtime entity** of a class and is **essential to call variables and methods of the class**.

When we want to access methods and variables from another class, we have to create an object of the class whose methods and variables we want to access.

Guiding questions

1. What are the different ways to create objects in java?
2. How can one create an object in java without using the new keyword?
3. How can one create an object in java without calling the constructor?

Different ways to create objects in Java

To create an object, Java provides various ways that we are going to discuss in this section, including:-

1. new keyword
2. newInstance() of the Class class
3. newInstance() of the Constructor
4. clone() method
5. Deserialization

1 new Keyword

creating objects using **new keyword** is very popular and common in java. Using this method user or system defined default constructor is called in order to initialize instance variables.

The new keyword creates a memory area in heap to store created object.

1. new Keyword+

Example:

In this example, we are creating two classes that is: - nkw and waysToCreateObjects. The later will be accessing the variables and methods of nkw calss. Lets begin with nkw class.

```
package arrayers;
public class nkw{
String b = "KUMU";
public void displayer() {
    System.out.println(" is Kumi University in
full");
}
}
```

1. new Keyword++

Example:

In this example, we are creating two classes that is: - nkw and waysToCreateObjects. The later will be accessing the variables and methods of nkw calss. Lets begin with nkw class.

```
package arrayers;
public class waysToCreateObjects {
    public static void main(String[] args) {
         nkw ob = new nkw();
        System.out.print(ob.b);
        ob.displayer();
    }
}
```

OUTPUT: KUMU is Kumi University in full.

Example 2.

Here we create two classes, that is Exams.java (with main method) and Student class (without main method but with a construtor)

We shall be using the Exams class to create Student class objects and run it.

Student Class

```
package exams;
public class Student {
    //student class property
    public int studentID = 200;
      //Constructor creation
    public Student(){
        /*To allow us see where the constructor is being called
            everytime this call is called
        */
        System.out.println("Student class Constructor called.");

    }
}
```

Exams Class

```
package exams;
public class Exams {
   public static void main(String[] args) {
        // Creation of Student class object using new keyword
        Student ob1 = new Student();

        //Printing out the student class object reference
        System.out.println(ob1);

        // Calling  Student class property before overriding
        System.out.println(ob1.studentID);
        //overriding student class property
        ob1.studentID =300;
        // printing  Student class property after overriding
        System.out.println(ob1.studentID);
   }}
```

OUTPUT

```
Student class Constructor called.
exams.Student@41629346
200
300
```

2. newInstance() of the **Class** class

In this case we will create an object using two methods called **newInstance()** and **forName()** which are found in the **Class** class. This **Class** class belongs to the java.lang.*; package.

To do this we will need to load our class called Student using the forName() method and call the newInstance() method. That is to say **Class.forName("exams.Student").newInstance();**

Note! Exams is also the package we are using in this examples.

Also note that class Loading **Class.forName("exams.Student").newInstance();** will lead to the need for exception handling therefore main() will have to throw the following exceptions: - ClassNotFoundException, InstantiationException, IllegalAccessException

Example

The second way to load a class is by directly using the name of the class being load dot class followed by a call to the newInstance() method. i.e. **Student.class.newInstance();**

So the two ways we could use to load the Student class are:

1. Class.forName("exams.Student").newInstance();

2. Student.class.newInstance();

To store the above however, we have to create the Student class object i.e.

1. Student ob2 = (Student)Class.forName("exams.Student").newInstance();

2. Student ob3 = Student.class.newInstance();

The good news is that the two options above dose the same operation.

Creating object using newInstance()

```
package exams;
public class Exams {
    public static void main(String[] args) throws ClassNotFoundException,
                                InstantiationException, IllegalAccessException{
        System.out.println("======================");
        //Create an object using newInstance() of Class class
        Student ob2 = (Student)Class.forName("exams.Student").newInstance();
        //Print object reference
        System.out.println(ob2);
        //Initialize the Student property to new value
        ob2.studentID=400;
        //Call student class property
        System.out.println(ob2.studentID);
    }
}
```

OUTPUT

```
======================
Student class Constructor called.
exams.Student@404b9385
400
```

3. Creating objects using the newInstance() of the **Constructor** class

The **Constructor** class belongs to java.lang.reflect.*; package. This class also provides us with a method called newInstance() which sounds exactly the same as the one in the **Class** class.

Before we use this method, lets first look at the difference between them.

Difference between Class.newInstance() and Constructor.newInstance()

Class.newInstance()

1. Can only invoke the no-argurments Constructor which is public by default.

2. Requires that the constructor should be visible.

3. Throws any exception thrown by the constructor

Constructor.newInstance()

1. Can invoke any parameter Inside the constructor.

2. Can also invoke private constructors in certain scenarios

3. Always wraps thrown exceptions with an invocationTargetException(wraps all exceptions into one)

Important Notes

Note!

1. When you create an object in java using the **Class** class newInstance() method, it calls internal Constructor to create an object. Therefore the **Constructor** newInstance() method is preferred over the **Class** class newInstance() method when it comes to creating objects.

2. Using Constructor newInstance() method to create an object is also called **reflective object creation**.

Creating Object using the constructor newIntance() method.

```
package exams;
public class Exams {
    public static void main(String[] args) throws NoSuchMethodException,
                                IllegalArgumentException, InvocationTargetException {
        System.out.println("=======================");
        //Create object using the Constructor newInstance() method
        Constructor<Student> constr = Student.class.getConstructor();
        Student ob4 = constr.newInstance();
        //Print the object reference
         System.out.println(ob4);
        //Initalice the Student property and print
        ob4.studentID = 600;
         System.out.println(ob4.studentID);
}
}
```

OUTPUT

```
=======================
Student class Constructor called.
exams.Student@6d311334
600
```

3. Clone() method

Cloning is the act of copying a given object properties to another object. In Java, clone() method is called an object. When a clone() method is called JVM creates a new object and then copy all the content of the old object into it.

When an object is created using the clone() method, a constructor is not invoked. To use the clone() method in a program the class must implement the cloneable interface and then override the clone() method.

Example:

In this example, we are creating an object using clone() method.

Note. Rules for using clone() method.

For one to be able use the clone method, he or she has to:-

1. make sure his class implements the Cloneable interface.

   In this case, we have to make Student class implement Cloneable interface.

2. override the clone() method.

   we will override the clone() method from inside our Student Class.

Example of an object copying properties and another using clone method.

```
Class awobi{
String Hair, Face, Eyes;
public static void main(String args[]){
//creating class object
awobi ob = new awobi();
/*setting properties using object*/
ob.Hair = “black”;
ob.Face = “long”;
ob.Eyes = “brown”;
```

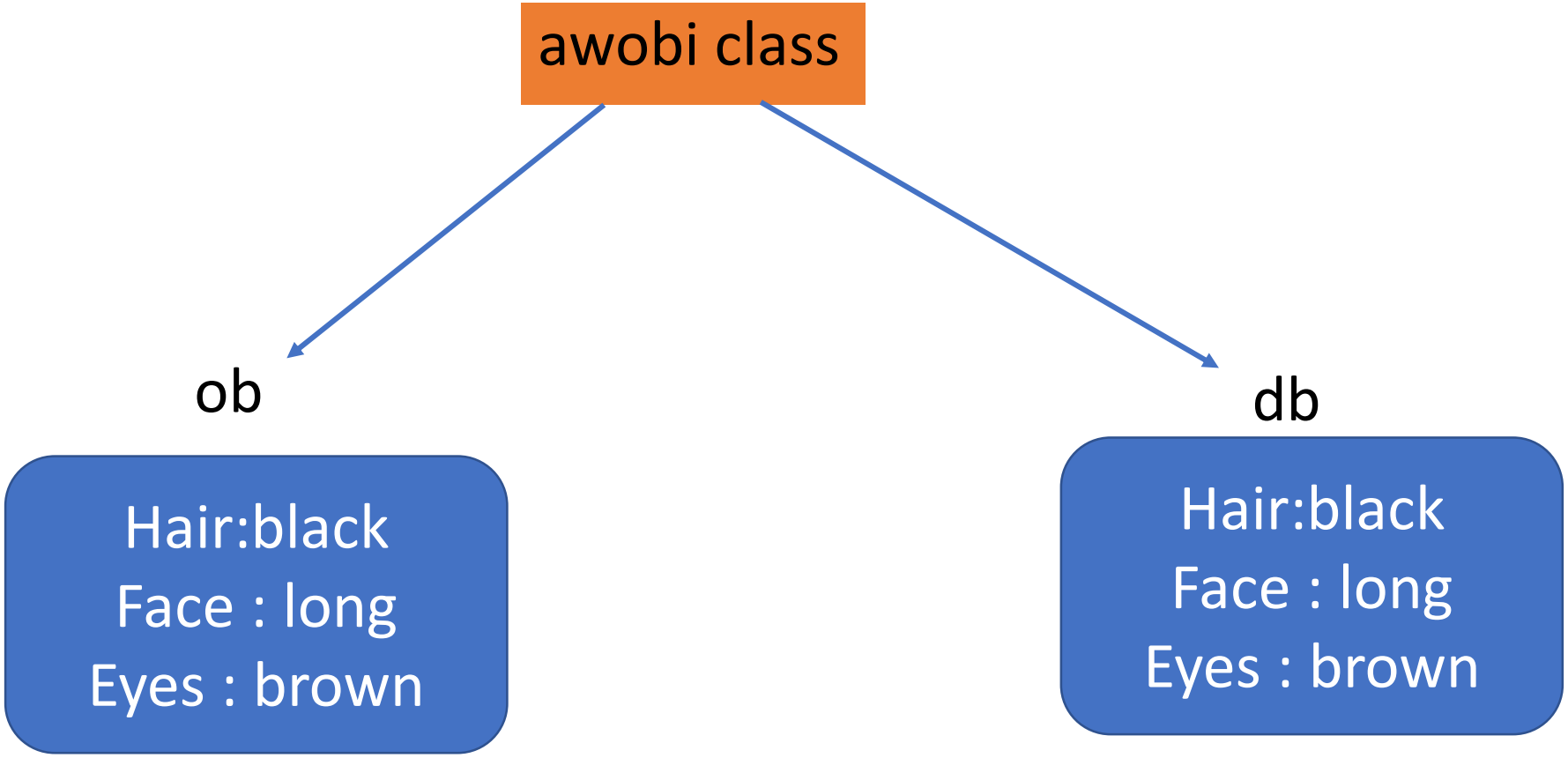
Example+

```
/*Create a separate object and copy the
properties of ob to it.*/
awobi db = ob.clone();


System.out.println(db);


}
}
```

Two Objects having same details



Object Creation using Clone() method

```
package exams;
public class Student implements Cloneable{
    //student class property
    public int studentID = 200;
         //Constructor creation
    public Student(){
        /*To allow us see where the constructor is being called
            everytime this call is called
        */
        System.out.println("Student class Constructor called.");
    }
//overriding the clone method.
    @Override
    protected Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Object Creation using Clone() method

```
package exams;
public class Exams {
   public static void main(String[] args) throws NoSuchMethodException,
                              IllegalArgumentException, InvocationTargetException
                                                          CloneNotSupportedException {

       Constructor<Student> constr = Student.class.getConstructor();
        Student ob4 = constr.newInstance();
       ob4.studentID = 600;

       //Create an object using clone() method e.g ob4
         Student ob5 = (Student) ob4.clone();
         //print object reference
         System.out.println(ob5);
         //Print objec value out
         System.out.println(ob5.studentID);
}
}
```

OUTPUT

```
==========================
exams.Student@682a0b20
600
```

4) deserialization

In Java, when an object is serialized and then deserialized, JVM create another separate object. When deserialization is performed JVM does not use any constructor for creating an object.

Example:

Lets see an example of creating object by deserialization concept. But before we create an object by deserialization we will need to do serialization first.

Note! if serialization code is not removed the program will call the constructor, however with the removal of the same, the constructor will not be invoked.

The class being called must be serializable. E.g `class Student serializable{`

Object creation by Deserialization

```
package exams;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
public class Exams {
   public static void main(String[] args) throws
                     FileNotFoundException, IOException {
```

```
        //Serialization
            FileOutputStream fos = new FileOutputStream("text.tex");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            Student s = new Student();
            oos.writeObject(s);

              System.out.println("==========================");
            //Create object by Deserialization
            FileInputStream fis = new FileInputStream("text.text");
            ObjectInputStream ois = new ObjectInputStream(fis);

        }
}
```

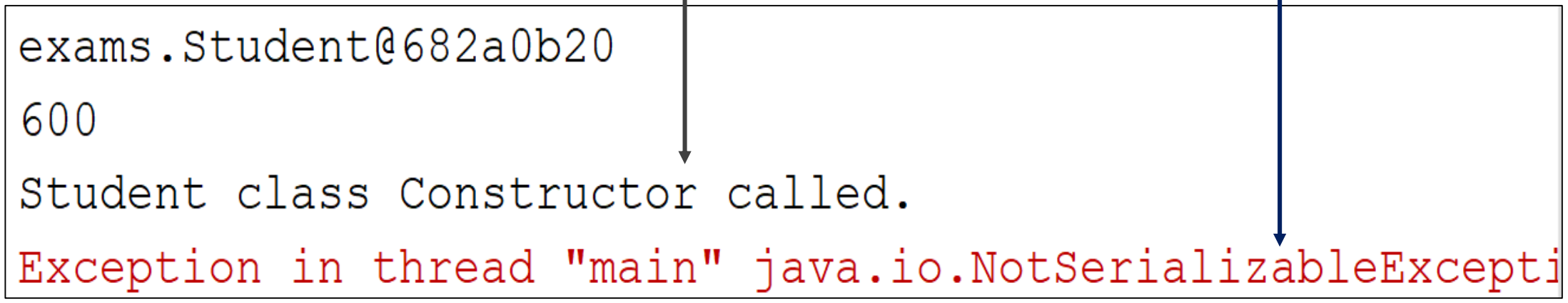
```
        //Creating the object
            Student ob7 =(Student)ois.readObject();
            //Print the object reference
            System.out.println(ob7);
        //Modify studentID value and Print
            ob7.studentID = 790;
            System.out.println(ob7.studentID);
}
}
```

Output

When the Student class is not serializable and the serialization code is not removed from the Exams main class.

Note two things: -

1. there will be NotSerializableException in main ……..,
2. the constructor will be called.



Output when serialization code is removed

When the Student class is serializable and the serialization code is removed from the main class.

Note.

1. No Constructor call
2. No Exception

We have seen the output without serializing the Student Class, lets serialize it.

```
package exams;
import java.io.Serializable;
public class Student implements Cloneable, serializable{
        //student class property
        public int studentID = 200;
        //Constructor creation
        public Student(){
        System.out.println("Student class Constructor called.");
    }
        //overriding the clone method.
    @Override
    protected Object clone() throws CloneNotSupportedException{
        return super.clone();
    }
}
```

Serialization code removed

```
        //Creating the object
        Student ob7 =(Student)ois.readObject();
        //Print the object reference
        System.out.println(ob7);
        //Modify studentID value and Print
        ob7.studentID = 790;
        System.out.println(ob7.studentID);
}
}
```

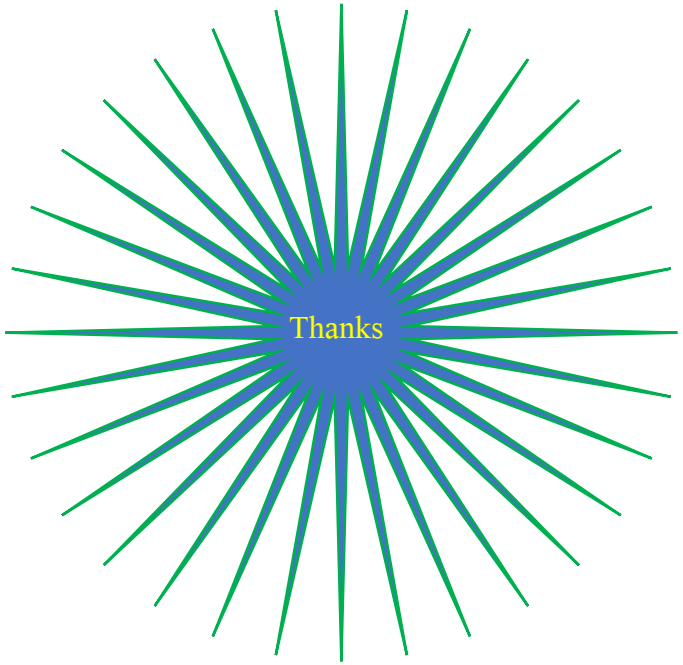
OUTPUT

```
==========================
exams.Student@14514713
790
```

Summary

1. Array in Java
    i. Types of arrays, declaration and initialization of arrays, how to print array elements out
2. Ways to create object
    i. Looked at various ways of creating objects in java including(using new keyword, using newInstance() from Class and Constructor classes), cloning and deserialization.

Thank you for Listening



Reference

Abhilash. (2017, October 17). *Object cloning in Java || different ways to create an object || part 4 || Clone( )*. YouTube. Retrieved June 3, 2022, from https://www.youtube.com/watch?v=CVQxs-zsUWg

Java arrays. Studytonight.com. (n.d.). Retrieved May 3, 2022, from https://www.studytonight.com/java/array.php