# Fundamentals of Object Oriented Programming

***CSN- 103***

**Dr. R. Balasubramanian**

**Associate Professor**

**Department of Computer Science and Engineering**

**Indian Institute of Technology Roorkee**

**Roorkee 247 667**

*balarfcs@iitr.ac.in*

*https://sites.google.com/site/balaiitr/*

## How can an object be unreferenced?

- By making the reference Null
- By assigning a reference to another
- By anonymous object etc.

## By nulling a reference:

```
Employee e=new Employee();
e=null;
```

## By assigning a reference to another:

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by e1 is
     //available for garbage collection
```

## By anonymous object:

```
new Employee();
```

`finalize()` method

`gc()` method

## finalize() method

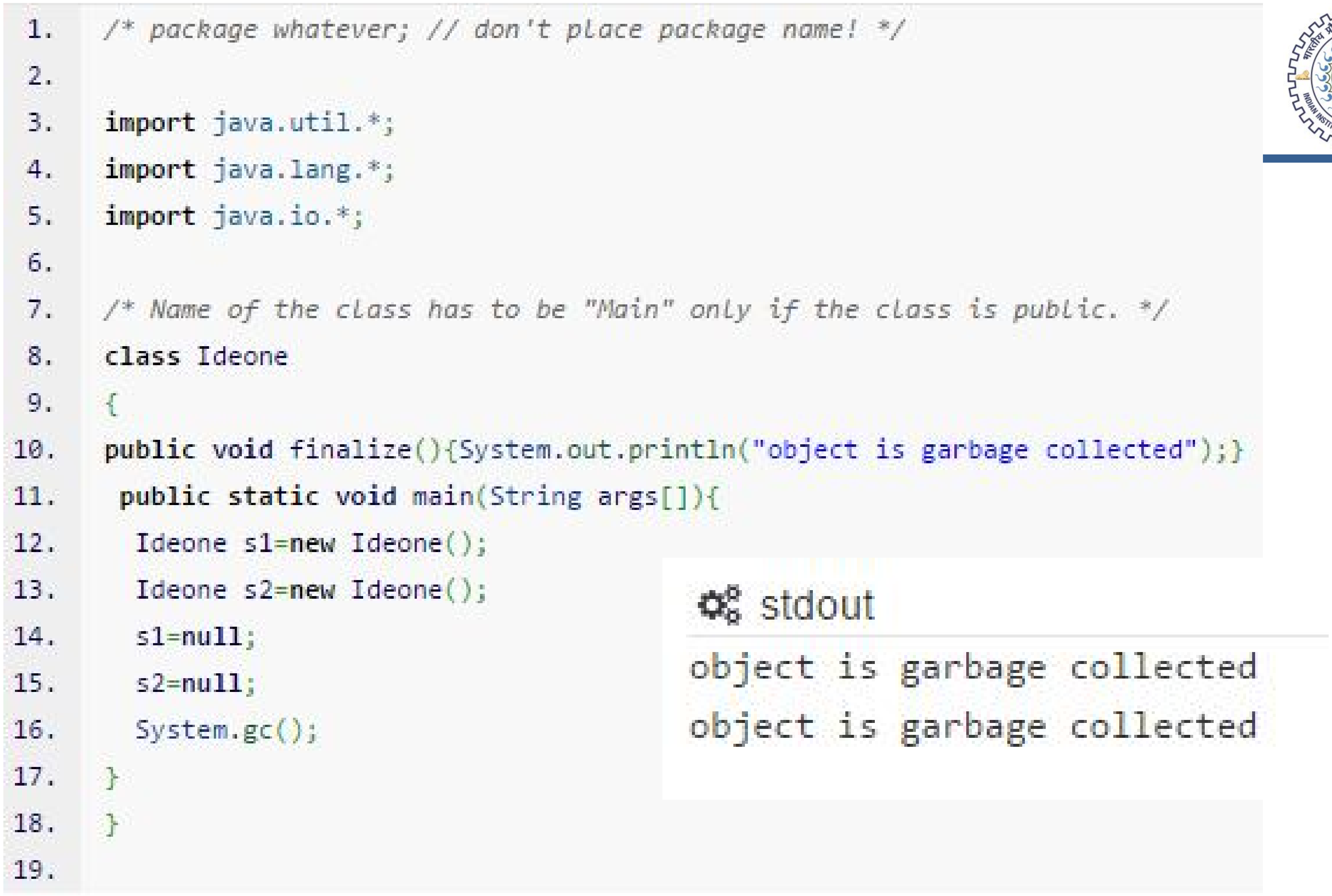
- The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
public void finalize(){}
```

## `gc()` method

- The `gc()` method is used to invoke the garbage collector to perform cleanup processing. The `gc()` is found in System and Run-time classes.

```
public static void gc(){}
```

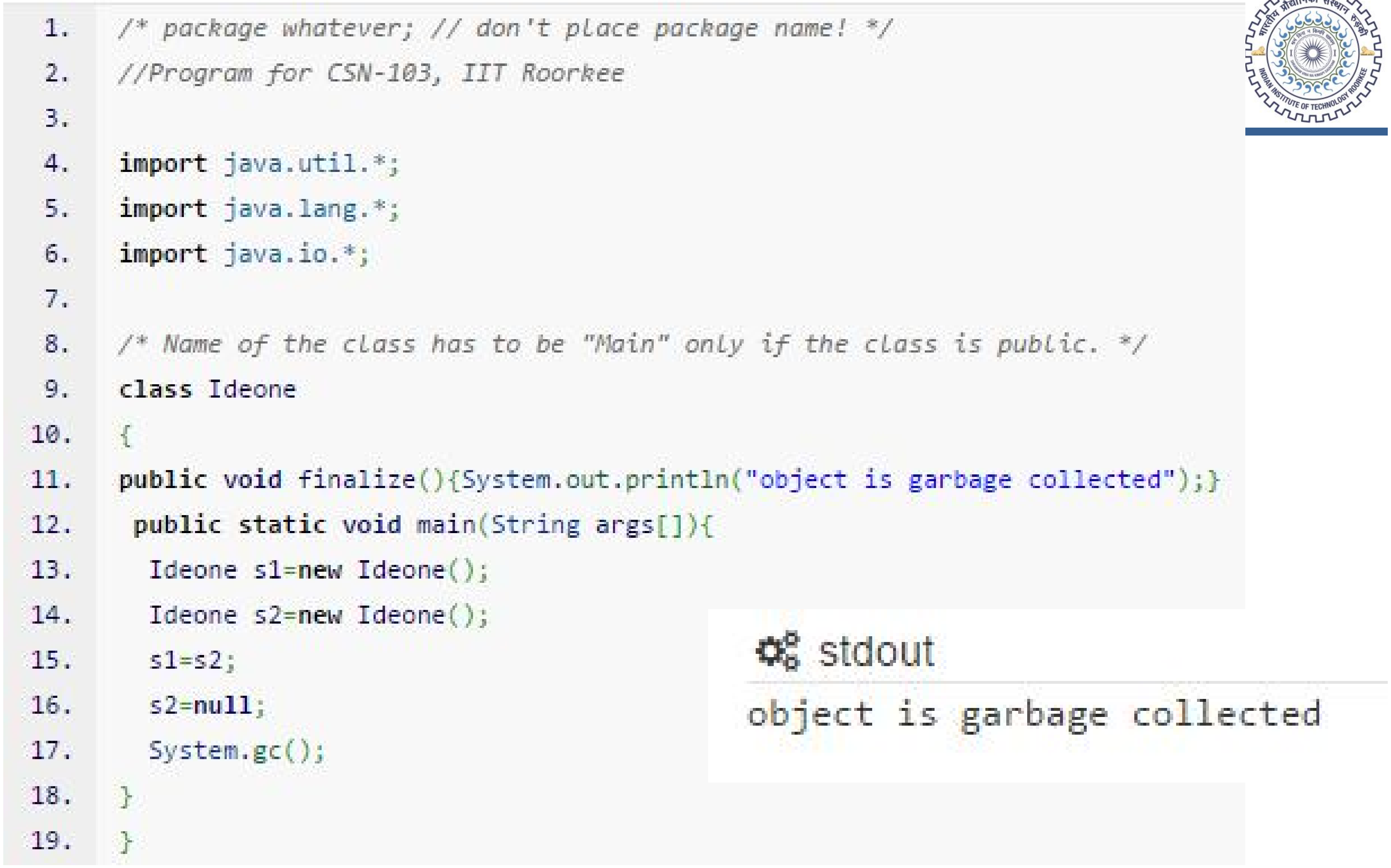


```java
/* package whatever; // don't place package name! */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Ideone
{
public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  Ideone s1=new Ideone();
  Ideone s2=new Ideone();
  s1=null;
  s2=null;
  System.gc();
}
}

```

stdout

```
object is garbage collected
object is garbage collected
```

https://ideone.com/dN2zOU

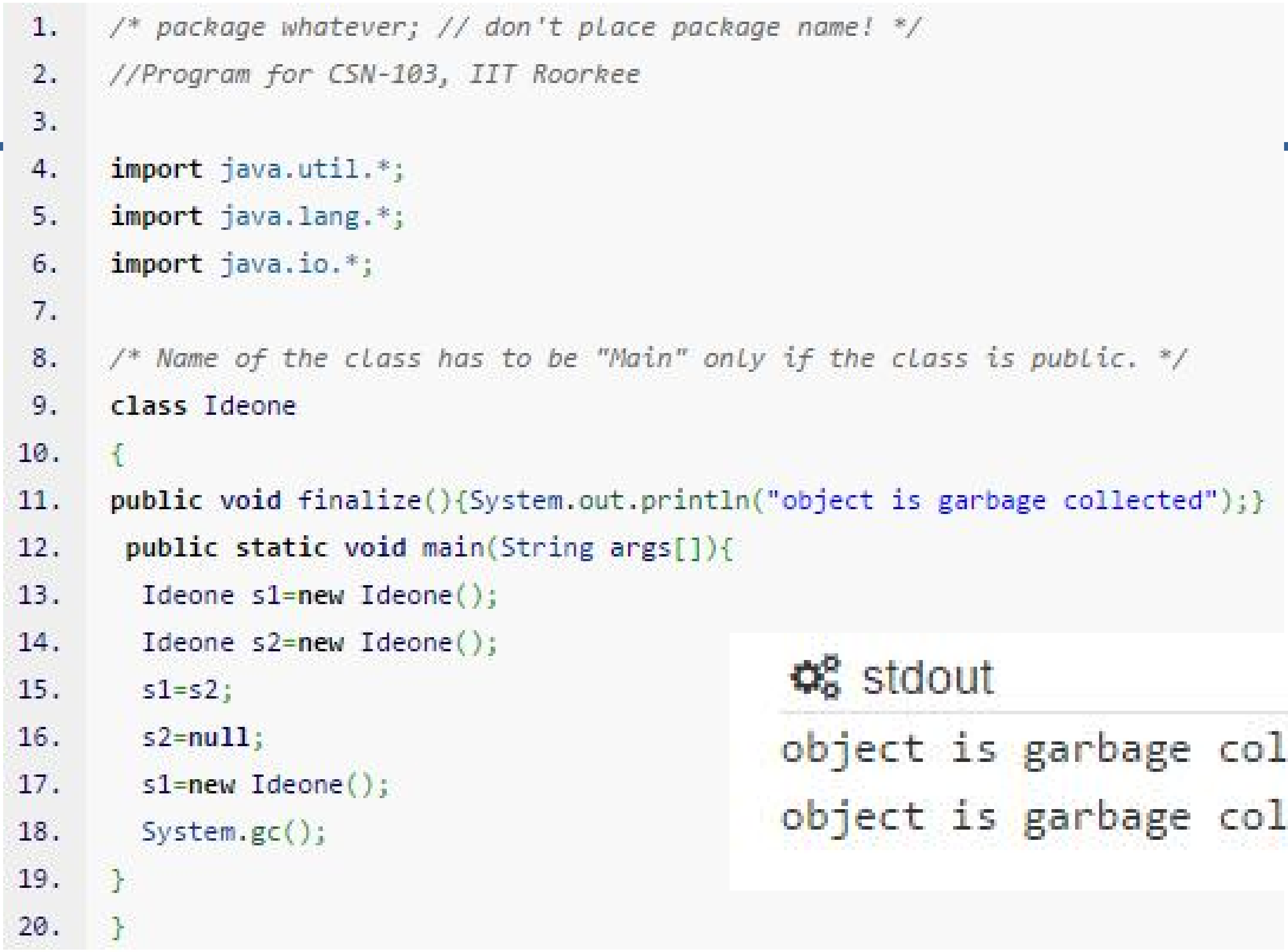


```
/* package whatever; // don't place package name! */
//Program for CSN-103, IIT Roorkee

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Ideone
{
public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  Ideone s1=new Ideone();
  Ideone s2=new Ideone();
  s1=s2;
  s2=null;
  System.gc();
}
}
```

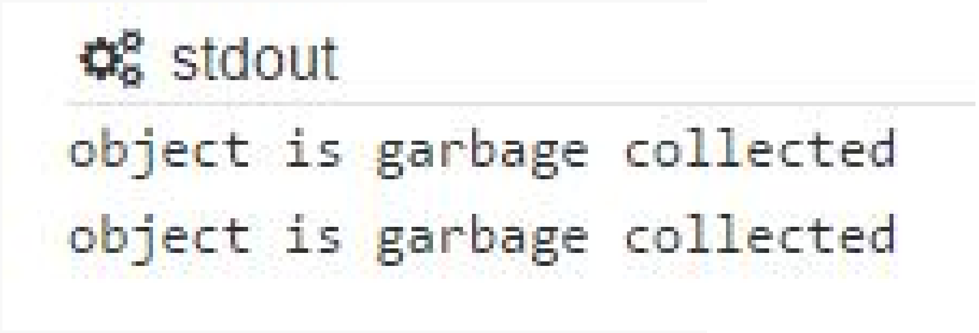
stdout

```
object is garbage collected
```

- https://ideone.com/pa48VC



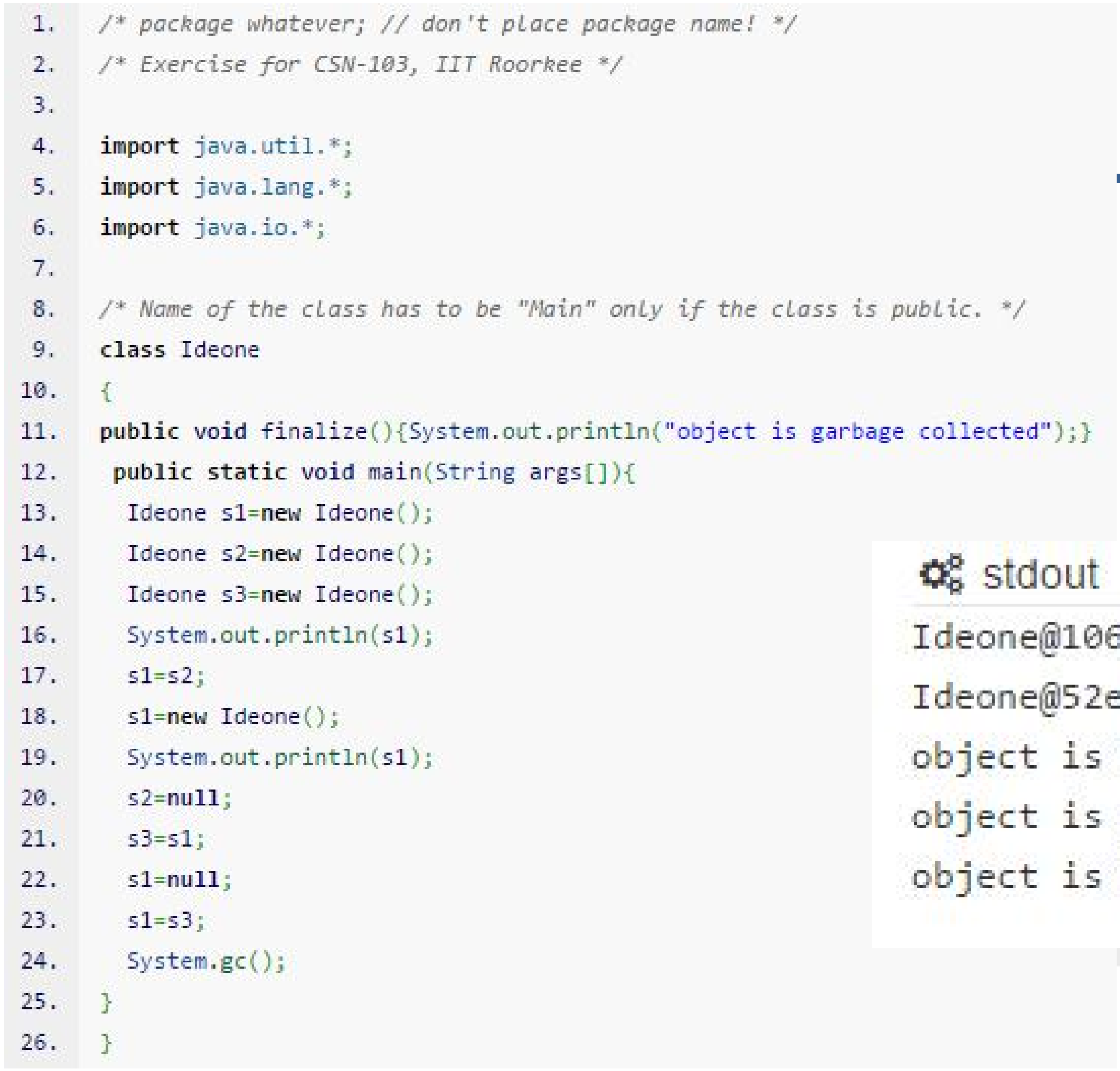
```java
/* package whatever; // don't place package name! */
//Program for CSN-103, IIT Roorkee

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Ideone
{
public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  Ideone s1=new Ideone();
  Ideone s2=new Ideone();
  s1=s2;
  s2=null;
  s1=new Ideone();
  System.gc();
}
}
```



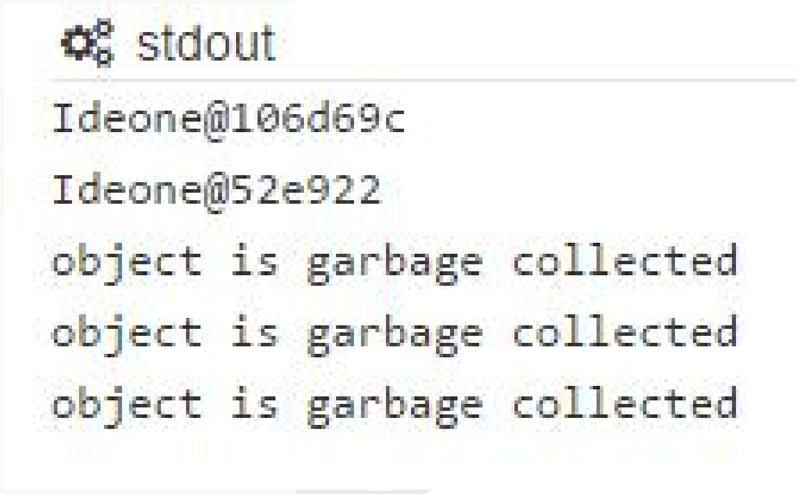
stdout

```
object is garbage collected
object is garbage collected
```

- https://ideone.com/BstbO6



```
/* package whatever; // don't place package name! */
/* Exercise for CSN-103, IIT Roorkee */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Name of the class has to be "Main" only if the class is public. */
class Ideone
{
public void finalize(){System.out.println("object is garbage collected");}
 public static void main(String args[]){
  Ideone s1=new Ideone();
  Ideone s2=new Ideone();
  Ideone s3=new Ideone();
  System.out.println(s1);
  s1=s2;
  s1=new Ideone();
  System.out.println(s1);
  s2=null;
  s3=s1;
  s1=null;
  s1=s3;
  System.gc();
}
}
```

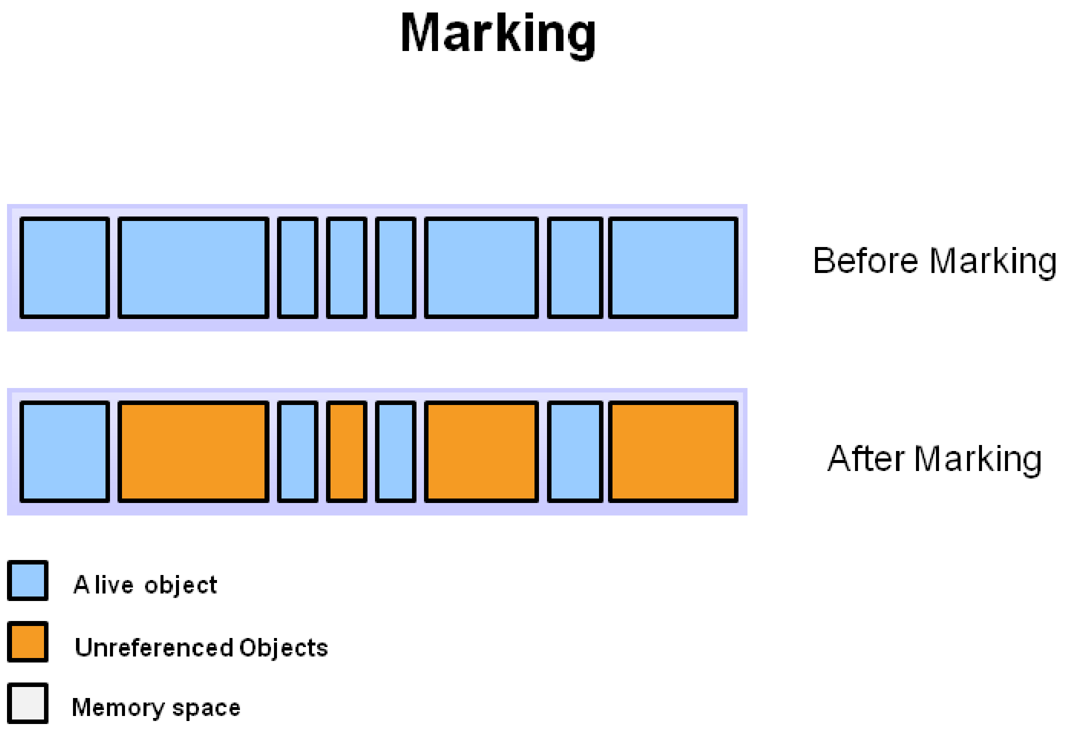


stdout

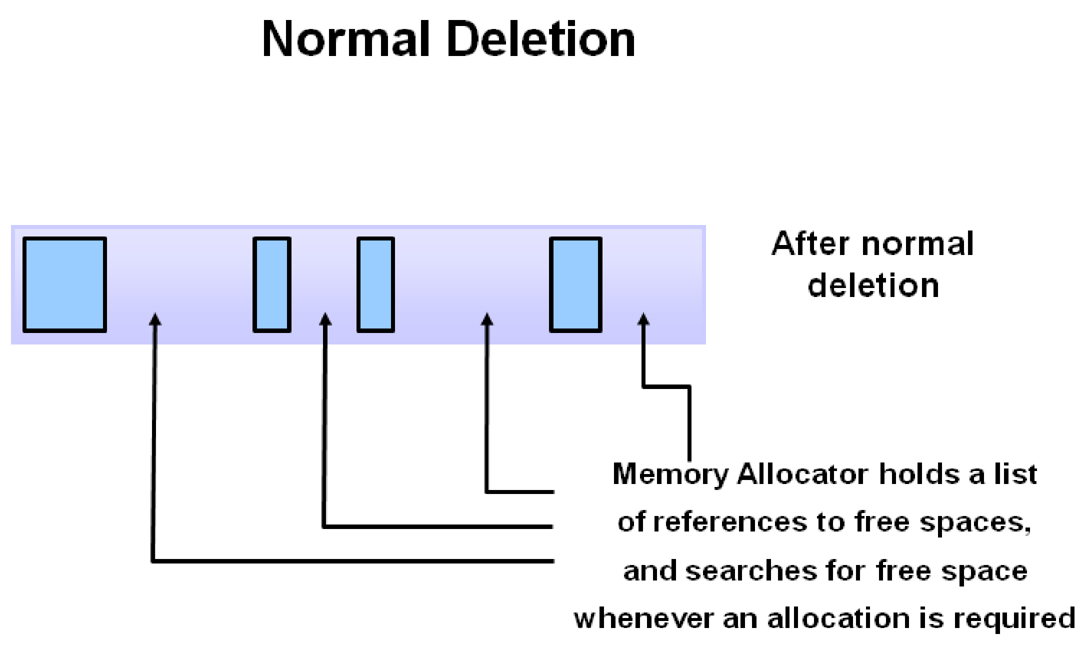
```
Ideone@106d69c
Ideone@52e922
object is garbage collected
object is garbage collected
object is garbage collected
```

- https://ideone.com/aD4PEs

## Marking

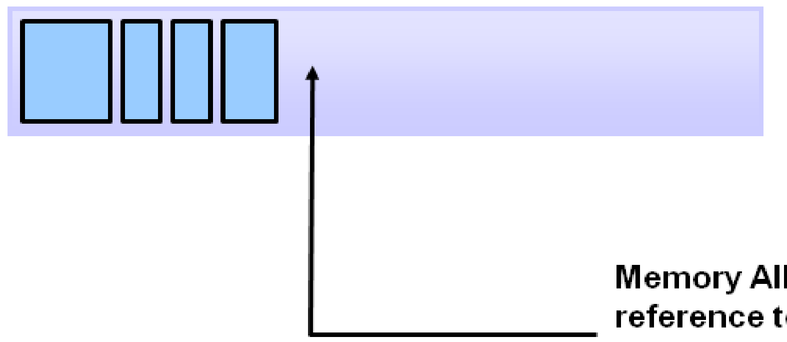

Marking
Before Marking
After Marking
A live object
Unreferenced Objects
Memory space

## Normal Deletion


Normal Deletion
After normal deletion
Memory Allocator holds a list of references to free spaces, and searches for free space whenever an allocation is required

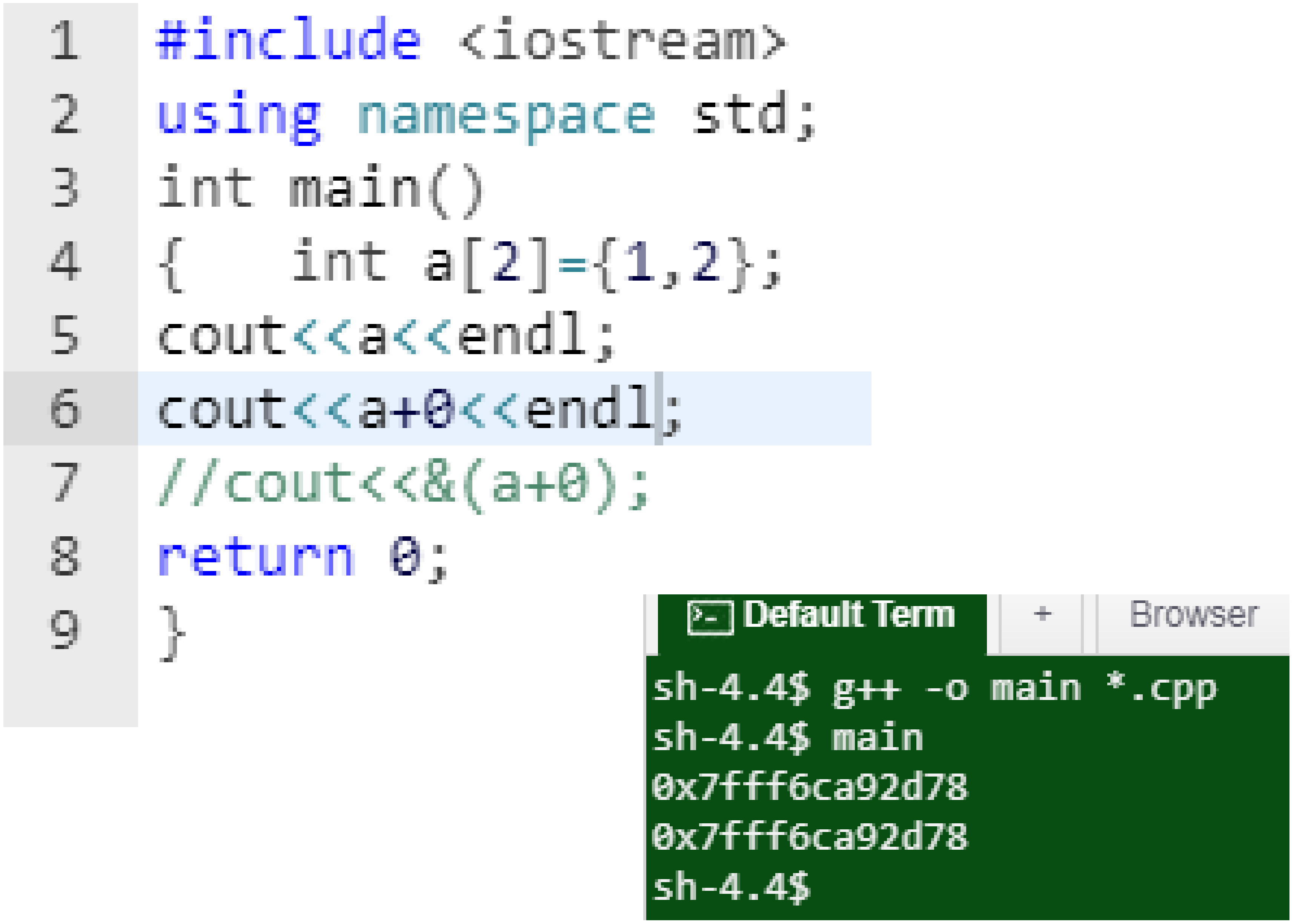
# Deletion with Compacting

Deletion with Compacting



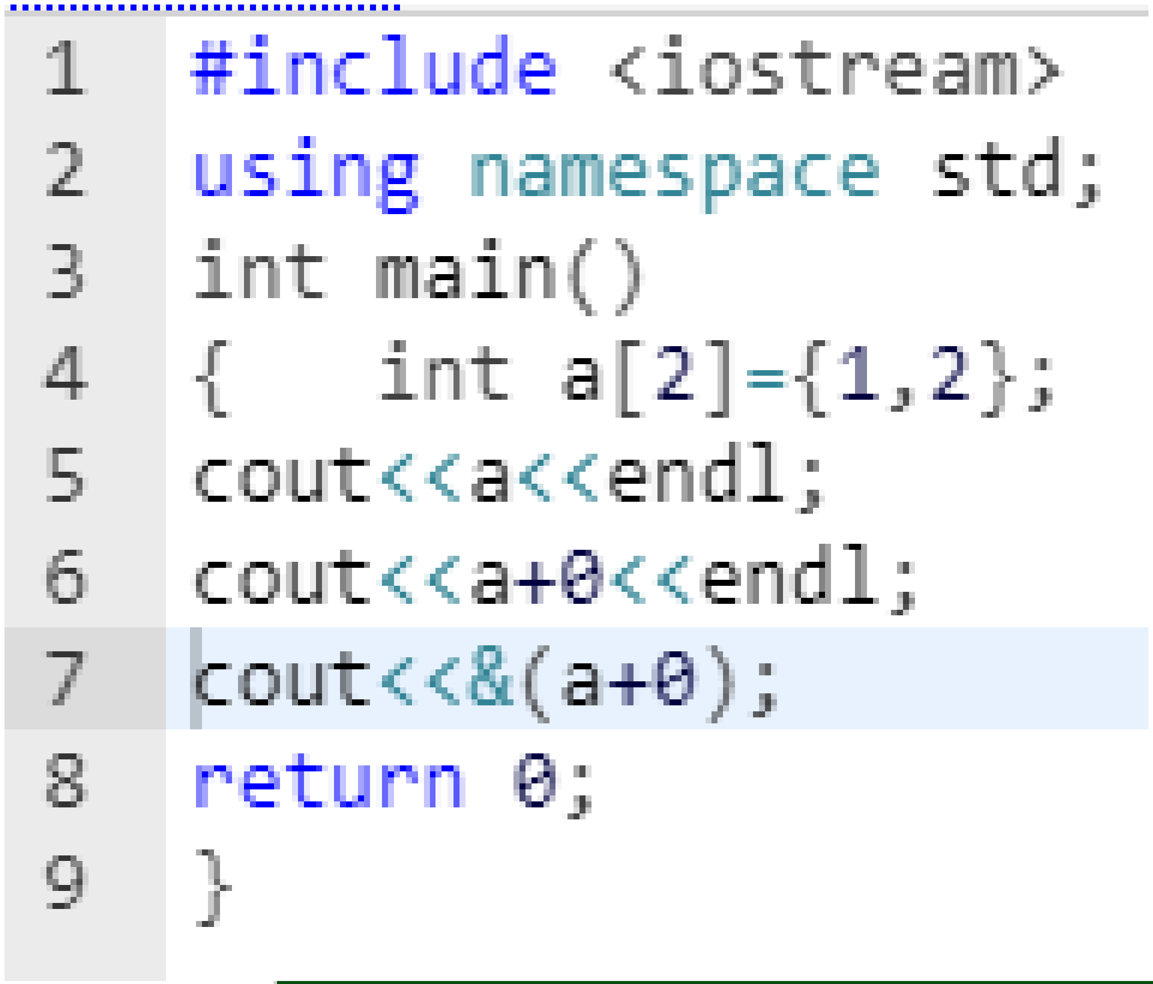
After normal Deletion with compacting

Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

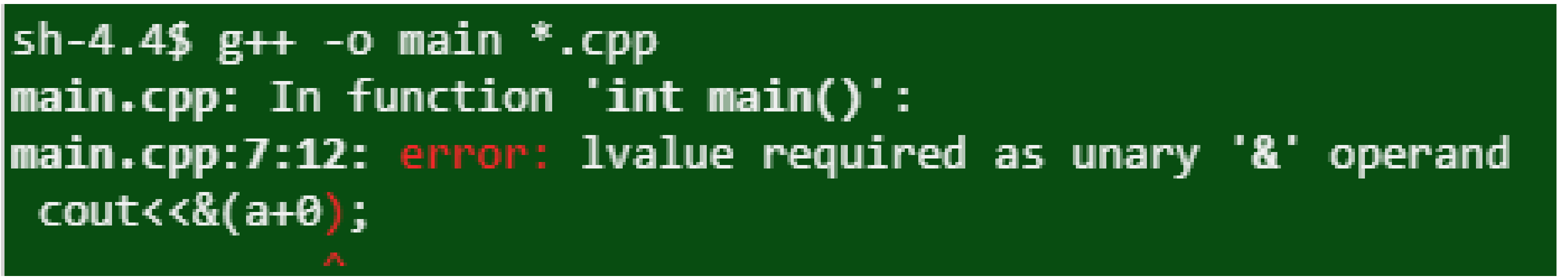


```cpp
#include <iostream>
using namespace std;
int main()
{   int a[2]={1,2};
cout<<a<<endl;
cout<<a+0<<endl;
//cout<<&(a+0);
return 0;
}
```

```
sh-4.4$ g++ -o main *.cpp
sh-4.4$ main
0x7fff6ca92d78
0x7fff6ca92d78
sh-4.4$
```

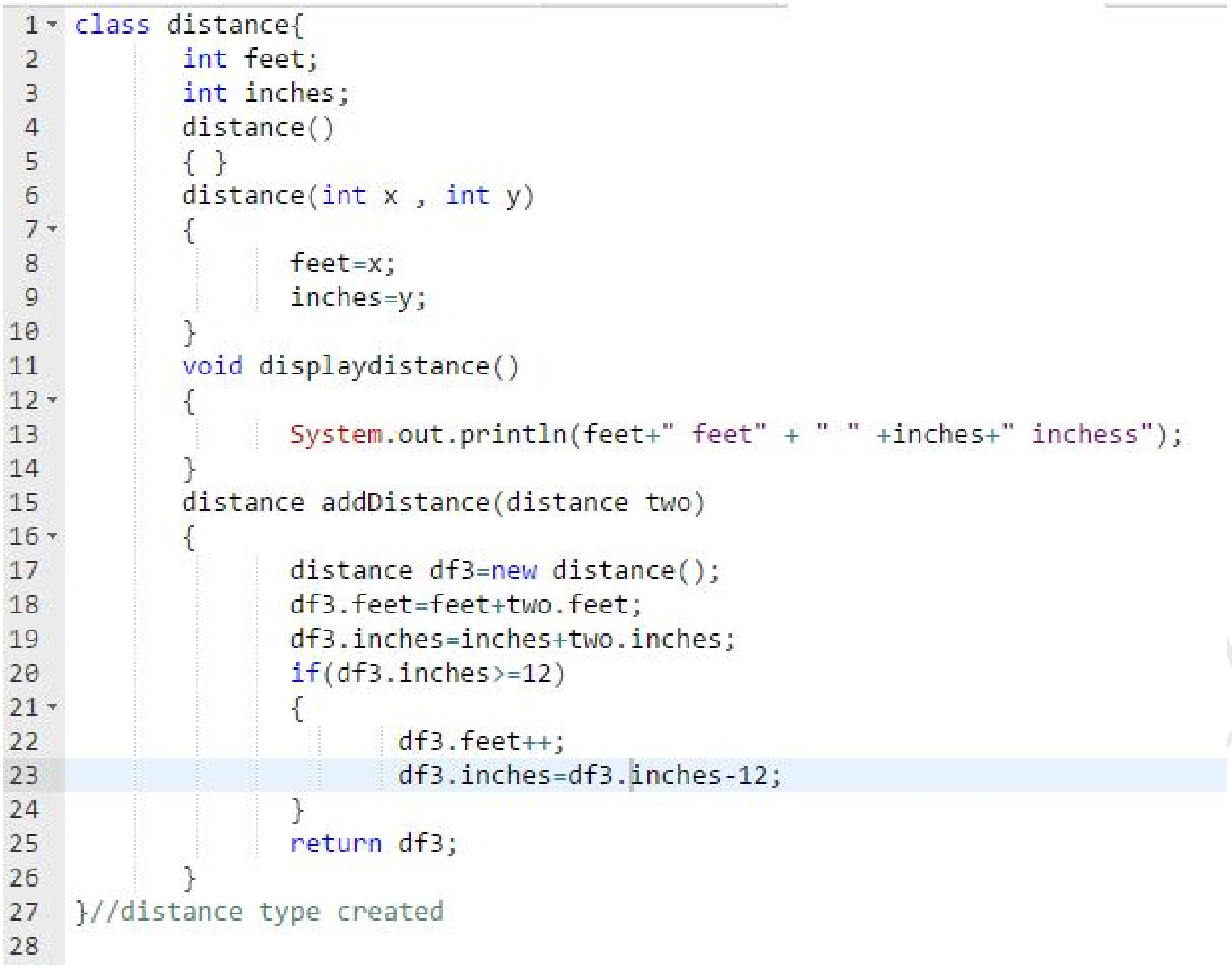


```cpp
#include <iostream>
using namespace std;
int main()
{    int a[2]={1,2};
cout<<a<<endl;
cout<<a+0<<endl;
cout<<&(a+0);
return 0;
}
```

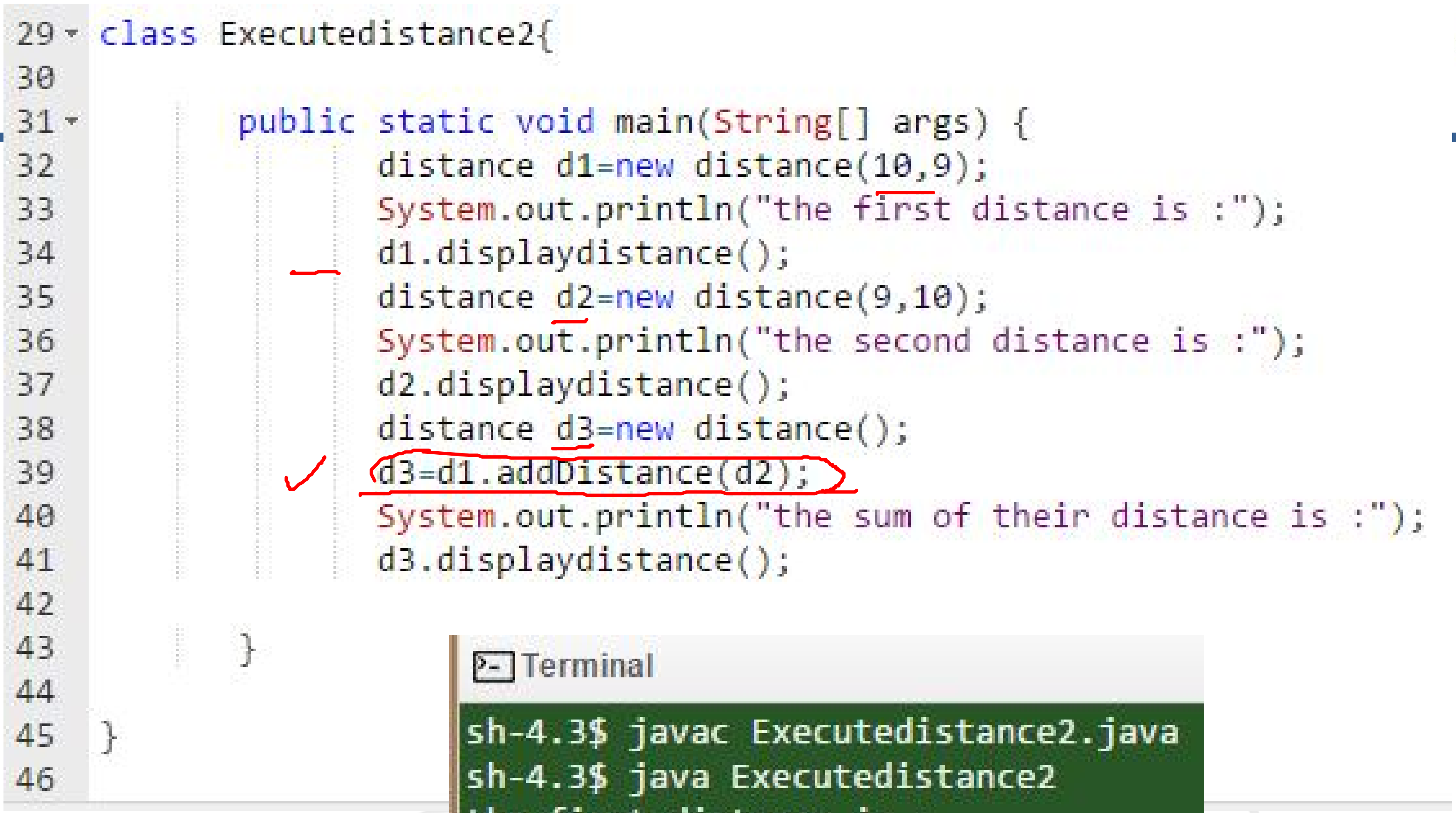


```
sh-4.4$ g++ -o main *.cpp
main.cpp: In function 'int main()':
main.cpp:7:12: error: lvalue required as unary '&' operand
 cout<<&(a+0);
            ^
```

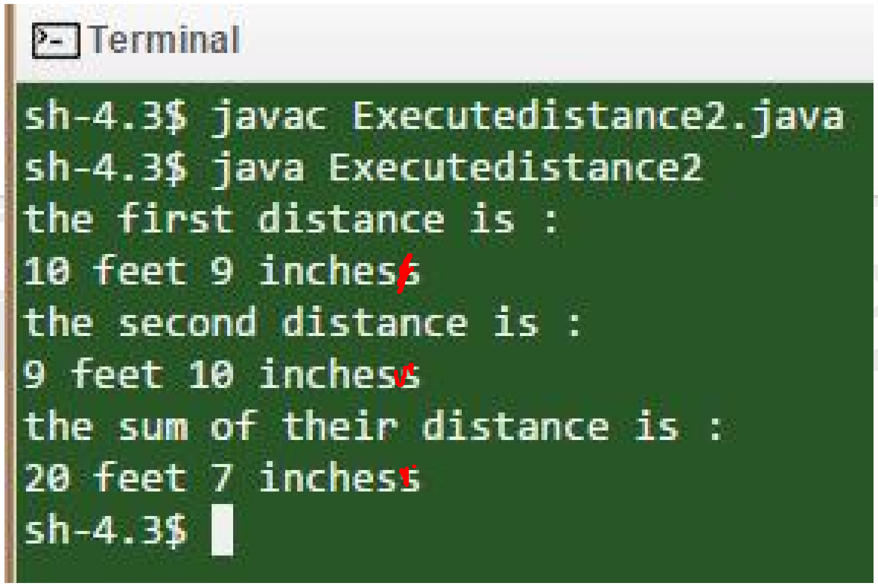
## Objects invoke methods



```
class distance{
        int feet;
        int inches;
        distance()
        { }
        distance(int x , int y)
        {
                feet=x;
                inches=y;
        }
        void displaydistance()
        {
                System.out.println(feet+" feet" + " " +inches+" inchess");
        }
        distance addDistance(distance two)
        {
                distance df3=new distance();
                df3.feet=feet+two.feet;
                df3.inches=inches+two.inches;
                if(df3.inches>=12)
                {
                        df3.feet++;
                        df3.inches=df3.inches-12;
                }
                return df3;
        }
}//distance type created
```

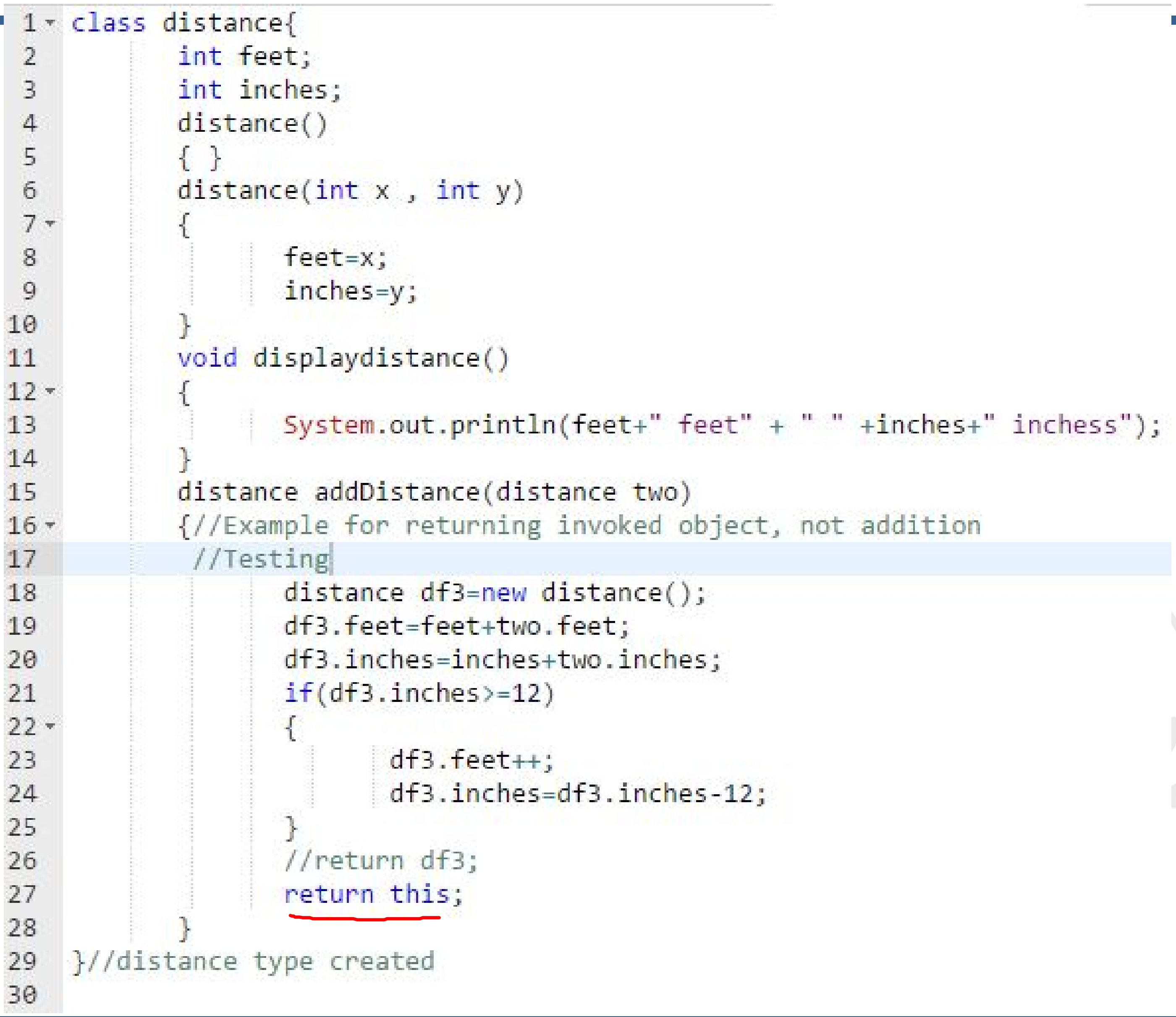


```java
class Executedistance2{

        public static void main(String[] args) {
                distance d1=new distance(10,9);
                System.out.println("the first distance is :");
                d1.displaydistance();
                distance d2=new distance(9,10);
                System.out.println("the second distance is :");
                d2.displaydistance();
                distance d3=new distance();
                d3=d1.addDistance(d2);
                System.out.println("the sum of their distance is :");
                d3.displaydistance();

        }

}
```

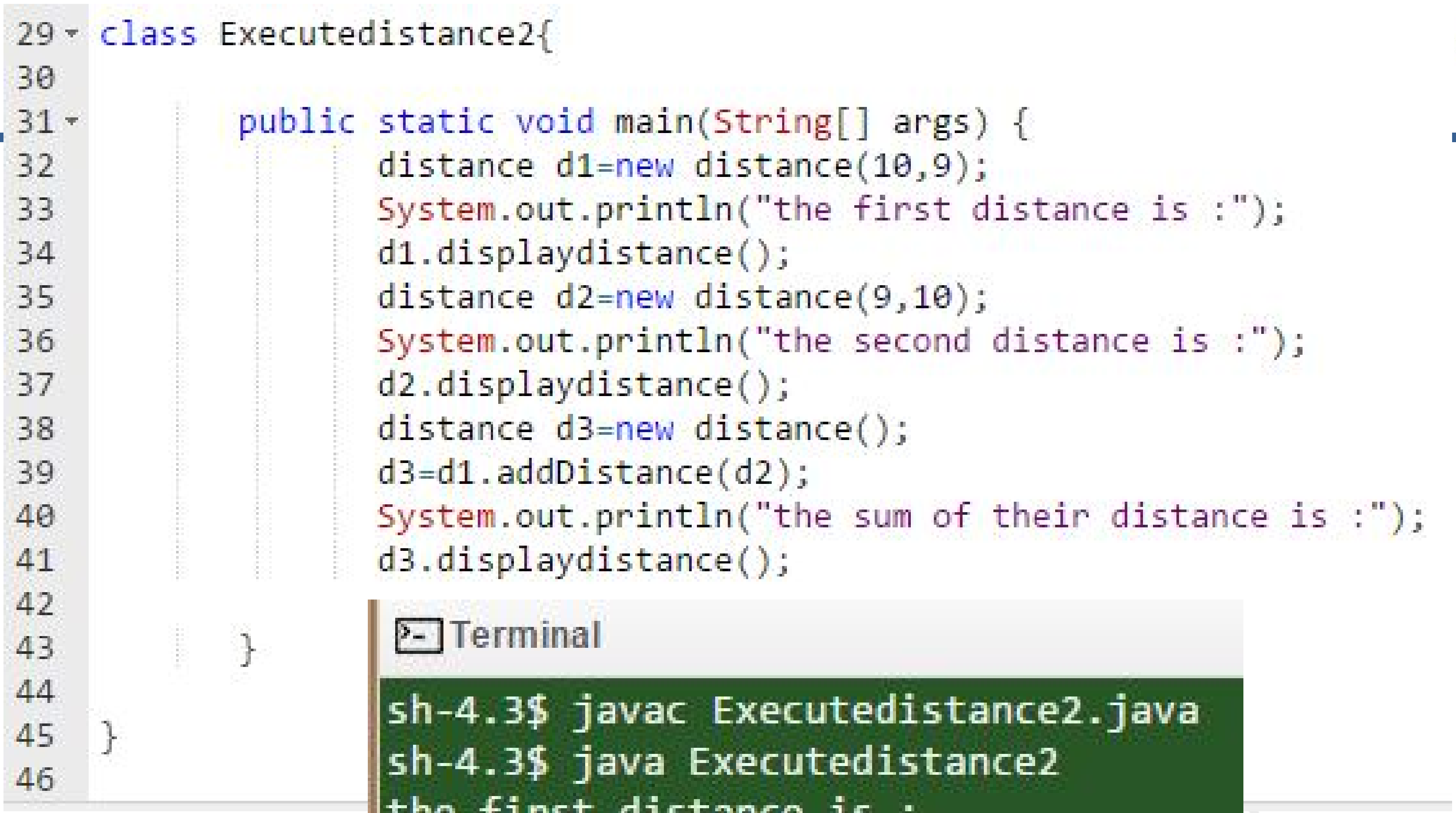


```
sh-4.3$ javac Executedistance2.java
sh-4.3$ java Executedistance2
the first distance is :
10 feet 9 inchess
the second distance is :
9 feet 10 inchess
the sum of their distance is :
20 feet 7 inchess
sh-4.3$
```

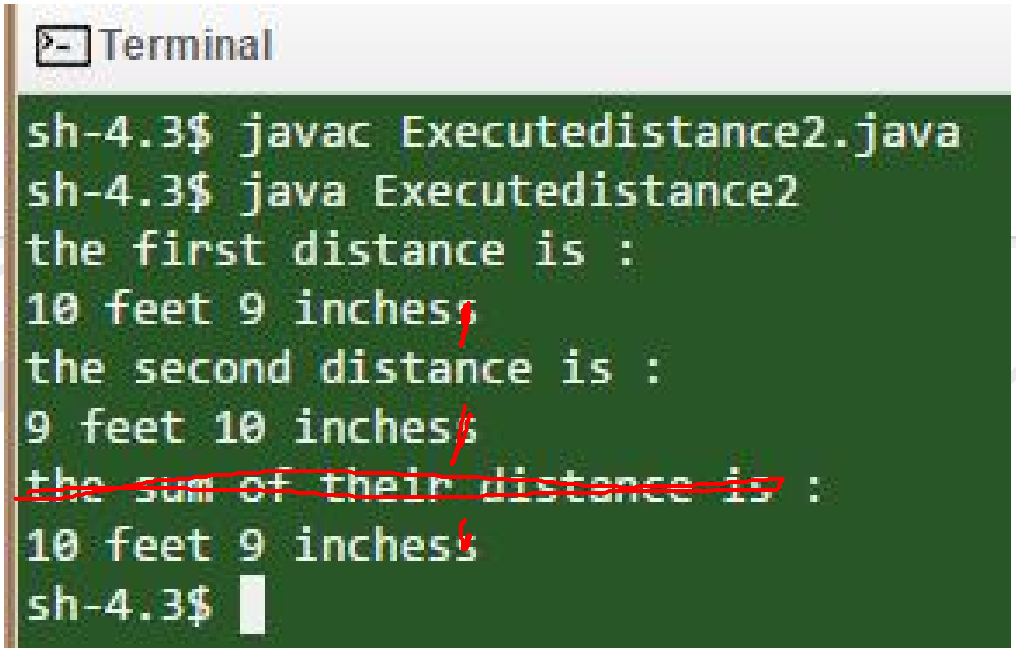
## Returning the invoked object



```
class distance{
        int feet;
        int inches;
        distance()
        { }
        distance(int x , int y)
        {
                feet=x;
                inches=y;
        }
        void displaydistance()
        {
                System.out.println(feet+" feet" + " " +inches+" inchess");
        }
        distance addDistance(distance two)
        {//Example for returning invoked object, not addition
         //Testing
                distance df3=new distance();
                df3.feet=feet+two.feet;
                df3.inches=inches+two.inches;
                if(df3.inches>=12)
                {
                        df3.feet++;
                        df3.inches=df3.inches-12;
                }
                //return df3;
                return this;
        }
}//distance type created
```



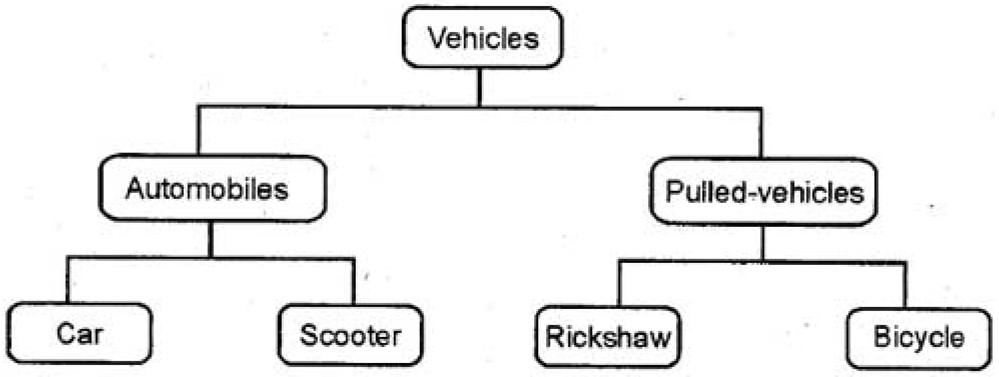
```java
class Executedistance2{

	public static void main(String[] args) {
		distance d1=new distance(10,9);
		System.out.println("the first distance is :");
		d1.displaydistance();
		distance d2=new distance(9,10);
		System.out.println("the second distance is :");
		d2.displaydistance();
		distance d3=new distance();
		d3=d1.addDistance(d2);
		System.out.println("the sum of their distance is :");
		d3.displaydistance();

	}

}
```



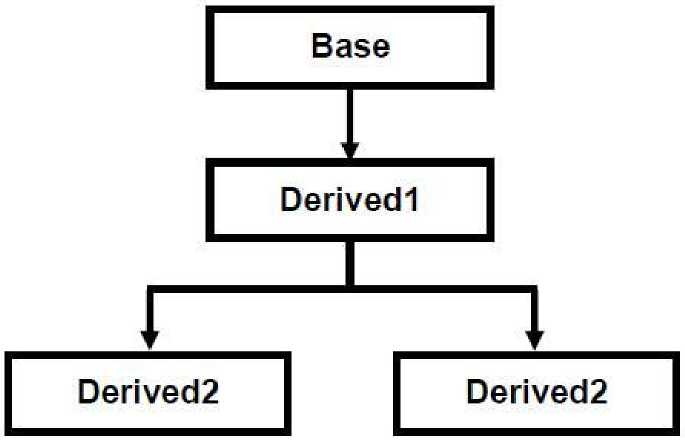
Terminal

```
sh-4.3$ javac Executedistance2.java
sh-4.3$ java Executedistance2
the first distance is :
10 feet 9 inches
the second distance is :
9 feet 10 inches
~~the sum of their distance is~~ :
10 feet 9 inches
sh-4.3$
```

## Inheritance



*Inheritance*



## Inheritance

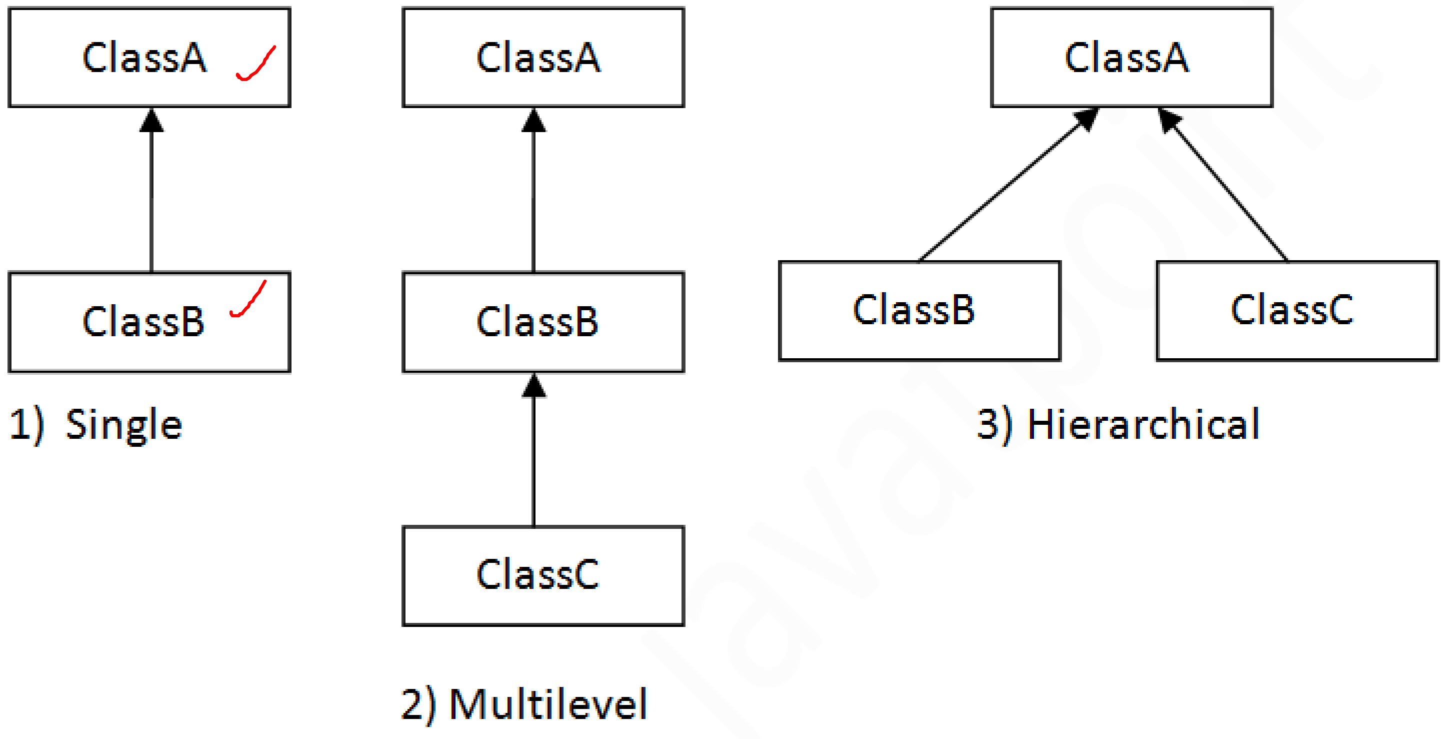
- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.
- With the use of inheritance the information is made manageable in a hierarchical order.

- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## Use of inheritance in java

- For Method Overriding (runtime polymorphism can be achieved).
- For Code Reusability.

## Types of Inheritance



## Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```