Introducing Classes, Objects and Methods

Introducing Classes, Objects and Methods

- Class Fundamentals
- How Objects are Created
- Reference Variables and Assignment
- Methods
- Returning from a Method
- Returning Value
- Using Parameters
- Constructors
- Parameterized Constructors
- The new operator Revisited
- Garbage Collection and Finalizers
- The this Keyword

Class Fundamentals

- A *class* is a template that defines the form of an object.

- It specifies both data and the code that will operate on that data.

- Java uses a class specification to construct **objects.**

- **Objects** are instances of a class.

- **A class** is essentially a set of plans that specify how to build an objects

```
class classname {
    // declare instance variables
    type var1;
    type varN;
        // declare methods
        type method1(parameters) {
        //body of method
        }
} //end class
```

Contd..

```
class Vehicle {
        int passengers; // number of passengers
        int fuelCap; // fuel capacity in litres
        int mpg; // fuel consumption in miles per litres
        }
```

- A class definition creates a new data type. In this case, the **new data type** is called **Vehicle.**

Types of class members

❑ **Types of class members**

 ➢ *Local variables* – inside method

 ➢ *Instance variables - common to class methods.*

 ➢ *Class variables – are static variables*

 ➢ *Reference variable - hold some reference*

Types of variables

```
Class Person {

    String  name;

    static int  count;

    void  fun ( ) {

        int data;

        //other code
    }

    public static void main (String [ ] args)
    {

        Person  P;
        P = new Person();

        P . fun( );

    }
}
```

Instance variable

Class variable

Local variable

Reference variable

Creates dynamic Object

How Objects are Created

- The following line is used to create object:

  Vehicle v =new Vehicle*( );*

  – *It declares variable called v of the class type Employee.*

  – *The declaration creates a physical copy of the object and assigns to v a reference to that object. This is done by using new operator.*

  – *the new keyword dynamically allocates memory for an object and returns a reference to it.*

Contd..

- Vehicle v; // declaring reference
- v = new Vehicle // allocating a vehicle object

  - The **dot operator(.)** links the name of an object with the name of a member.

  - The general form of the **dot operator** is
    object.member

  - In general the dot(.)operator is used to access **both instance variables and methods.**

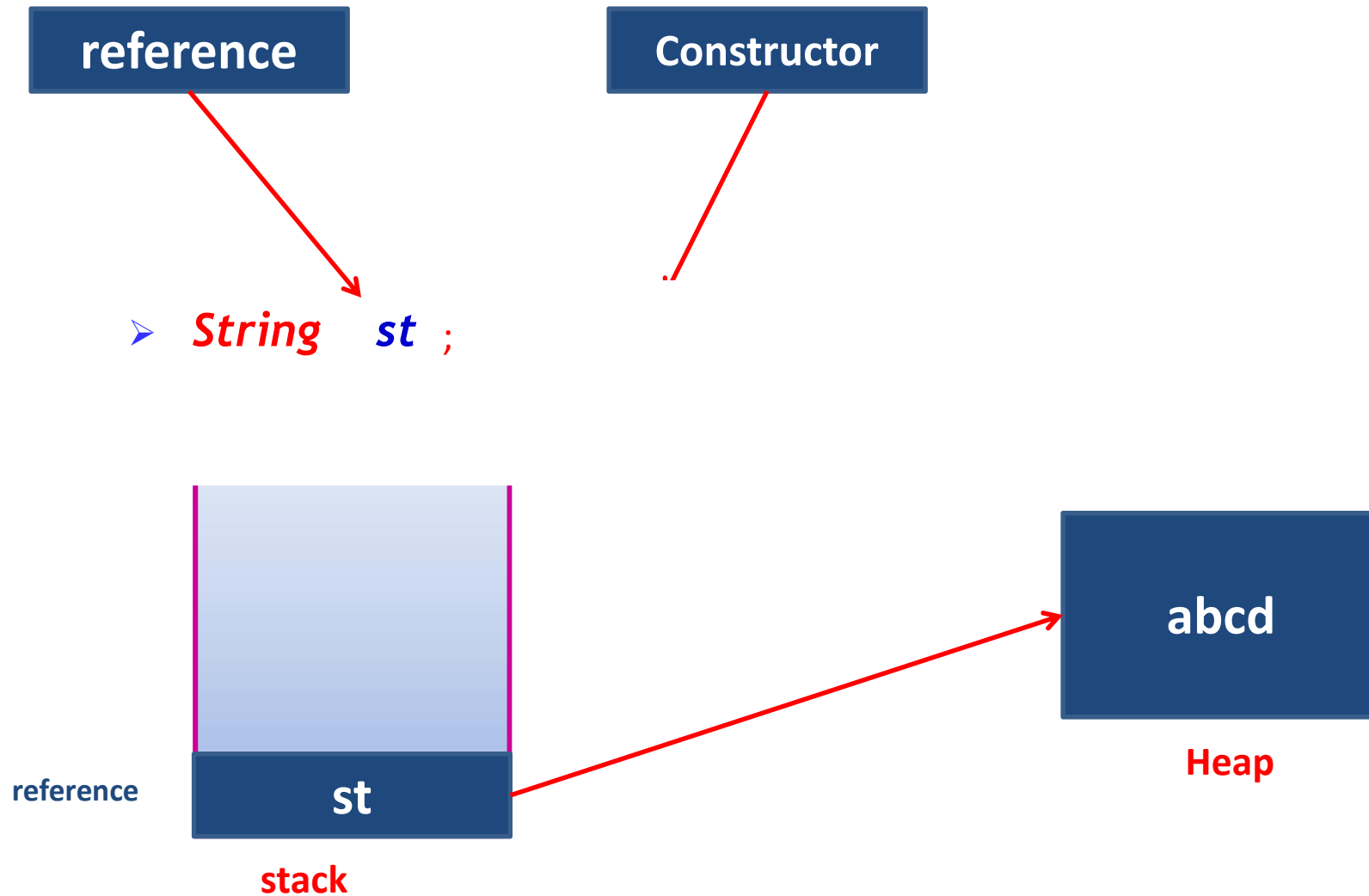# Creating object

reference

Constructor

➢ *String* *st* ;

reference

st

stack

abcd

Heap

What should be the filename?

```java
class Vehicle {
    int passengers;
    int fuelCap;
    int mpg;
}
class VehicleDemo {
    public static void main(String[] args) {
        Vehicle v = new Vehicle();
            int range;
                v.passengers = 7;
                v.fuelCap = 16;
                v.mpg = 21;
                range = v.fuelCap * v.mpg;
    System.out.println("Minivan can carry " + v.passengers + " with a range of
" + range); }
}
```

Reference variables and assignment

- Assigning one object reference variable to another.

> *Rectangle* r1 = new *Rectangle*( );
> *Rectangle* r2 = r1; // assigning r1

- r1 is reference variable which contain the address of Actual Rectangle Object
- r2 is another reference variable
- r2 is initialized with r1 means – "**r1 and r2**" both are referring same object , thus it does not create duplicate object , nor does it allocate extra memory.

Rectangle r1 = new Rectangle( );

r1

length

breadth

Rectangle object

Rectangle r2 =r1

r2
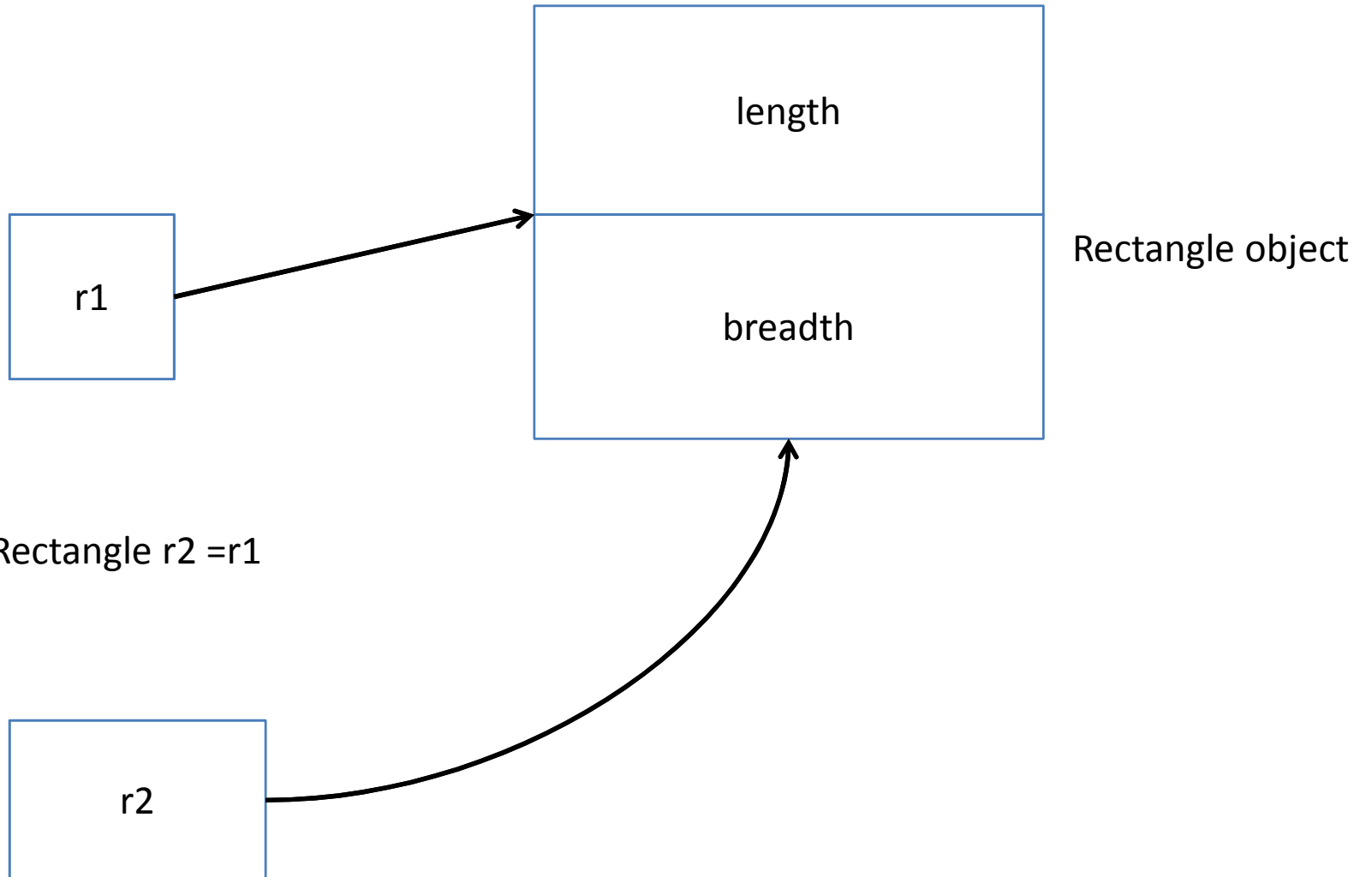
```java
class Rectangle {
   double length;
   double breadth;
}

class RectangleDemo {

   public static void main(String args[])  {

   Rectangle r1 = new Rectangle();
   Rectangle r2 = r1;

   r1.length = 10;
   r2.length = 20;

   System.out.println("Value of R1's Length : " + r1.length);
   System.out.println("Value of R2's Length : " + r2.length);

   }
}
```

Methods

- A method contains the statement that define its action.

  - Each method has a name
  - It is the same name used to call the method
  - General form

  re-type name(parameter-list)
   {
      //body of method
   }

Contd..

- *return_type* is nothing but the value to be returned to an calling method.

- *method_name* is an name of method that we are going to call through any method.

- *arg1,arg2,arg3* are the different parameters that we are going to pass to a method.

Contd..

- Return type of method

1. Method can return any type of value.

2. Method can return any Primitive data type.

   int sum(int a, int b);

3. Method can return Object of Class Type.

   Rectangle sum (**int** num1,int num2);

4. Method sometimes may not return value.

   void sum (**int** num1,unt num2);

Contd..

- ***Method Name***

1. Method name must be valid identifier.
2. All [Variable naming rules](#) are applicable for writing Method Name.

Parameter List

1. Method can accept ***any*** number of parameters.
2. Method can accept any ***data type*** as parameter.
3. Method can accept ***Object*** as Parameter
4. Method can accept ***no*** Parameter.
5. Parameters are separated by ***Comma***.
6. Parameter must have ***Data Type***

Returning from a method

```java
class Rectangle {
  double length;
  void setLength(int len)
    {
      length = len;
    }
}
class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();
  r1.setLength(20);

  System.out.println("After Function Length : " + r1.length);

  }
}
```

Returning a value

- Return values are used for a variety of purposes in programming.
- In some cases, such as **sqrt(),** the return value contains the outcome of some calculation.
- In other cases, the return value simply indicate success or failure.
- In still others, it may contain a status code.
- Methods return a value to the calling routine using this form of **return:**

  return value;

Contd...

```java
class Rectangle {
    double lenght, breadth;
void setLength(double len){
    length= len;
}
int getLength()
{
    return lenght*length;
}
```

```java
class RectDemo {
 public static void main(String [] args)
{
    Rectangle r1 = new Rectangle();
    r1. setlength(10);
    int sum = r1.getLenght();
    System.out.println("length"+sum
);
}
}
```
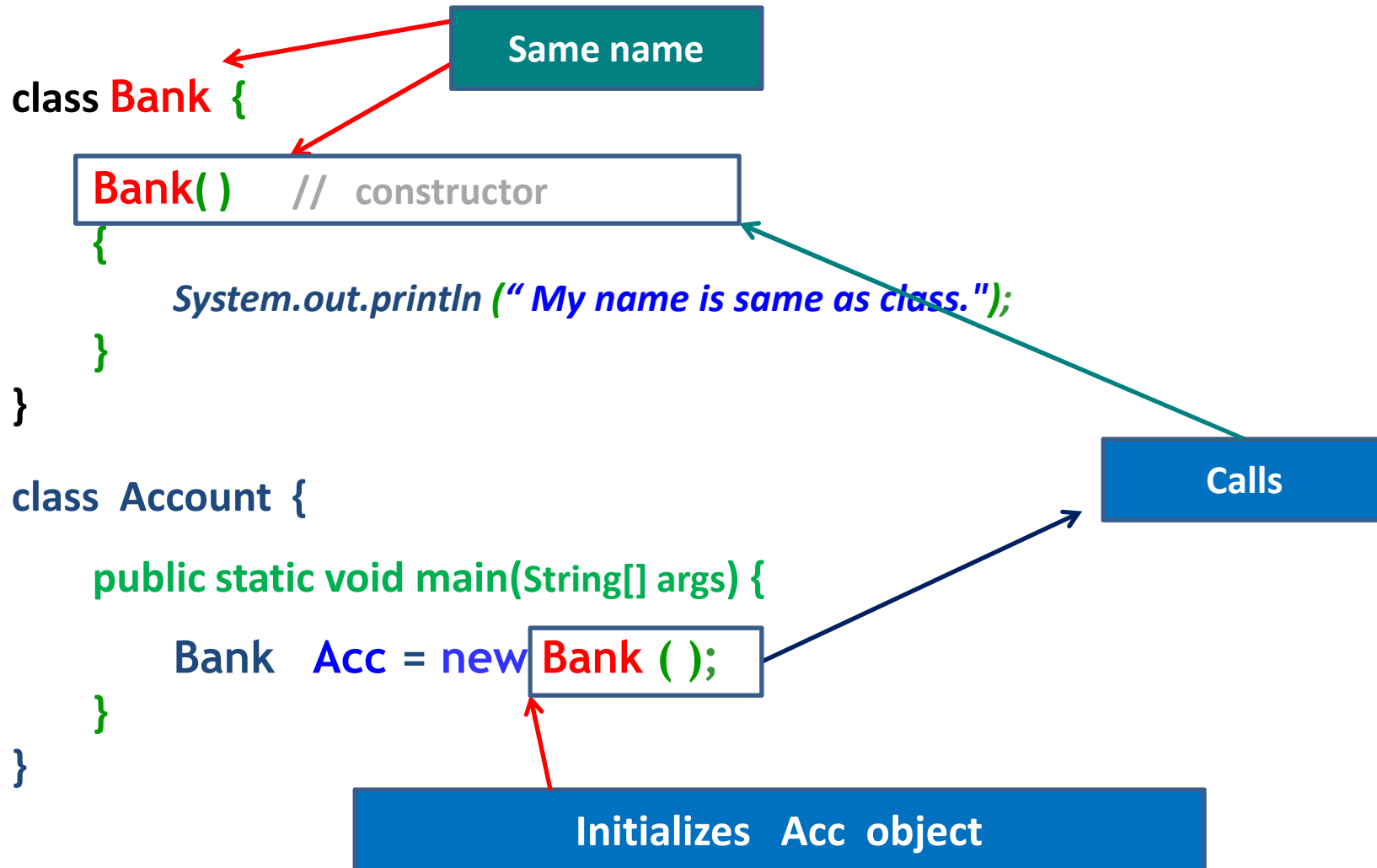
Using Parameters

- It is possible to pass one or more values to a method when the method is called.

- A value passed to a method is called an **argument.**

- Inside the method, the variable that receives the argument is called a **parameter.**

- Parameters are declared inside the parentheses that follow the method's name.

- The parameter declaration syntax is the same as that used for variables.

What is a constructor?

- a member function which initializes a class.

- A constructor has:

    (i) the same name as the class itself

    (ii) no return type

What is the name of the file

Same name

class Bank {

Bank( )   **// constructor**

{

    System.out.println *(" My name is same as class.");*

}

}

Calls

class Account {

    public static void main(String[] args) {

        Bank   Acc = new Bank ( );

    }

}

Initializes Acc object

```
class Animal {

    String name;

    boolean goodDog;

    Animal ( String Dname, boolean val )  {   // Parameterized constructor
    {

        name = Dname;

        goodDog = val;

    }
}

class Demo {
    public static void main ( String [] args)    {

    Animal   dog = new  Animal ( "Dobberman" , false ) ;

    // ..

    }
}
```

Calls

Destructor..

- A destructor is a special method typically used to perform cleanup after an object is no longer needed

- No destructors in java.

- Alternative in JAVA : garbage collection

About garbage collection..

- **Garbage Collection** is carried by a *daemon thread* called *Garbage Collector*.

- **Destroying object by garbage collector**

  - ✓ Before Destroy **invokes finalize () method.**

- **User can't force Garbage collection**; **JVM triggers it** if needed.

- **Requesting garbage collection**

  - ✓ `System.gc()` and `Runtime.gc()`

  - ✓ **send request to JVM** but **Garbage collection** is *not guaranteed.*

```
protected void finalize ( )
{
    //finalize code

}
```

- *A finalizer* in Java is the opposite of a constructor.

- a finalizer method performs finalization for the object.

- Garbage collector can't free resources, such as open files and network connections.

- you need to write a finalize() method for any object that needs to perform such tasks.

this – keyword

- *this* is a reference variable that refers to the *current object.*
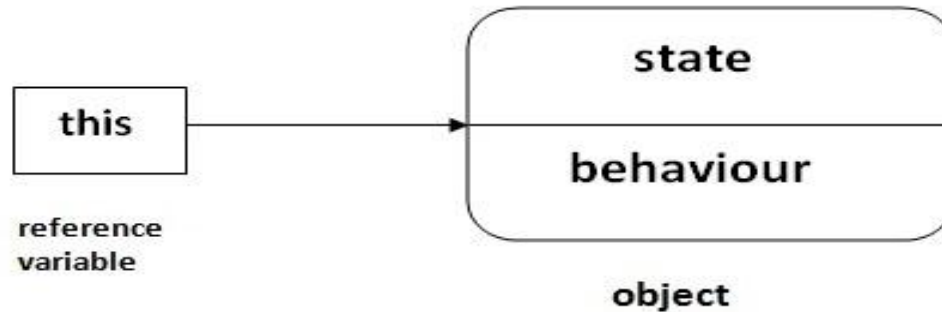
- 5 usage of this keyword.
  1. *this.variable*   :- access instance variable.
  2. *this.method()*   :- calls method of current class.
  3. *this()*   :- calls constructor of current class .
  4. *method(this)*   :- passes current object as an argument.
 5. *return this :-* returns instance of current class.

this – 5 uses

1. *this.variable* : to refer current class instance

```
//example of this keyword
class Student {
    int id;
    String name;

    Student (int id,String name){
    this.id = id;
    this.name = name;
    }
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[]){
    Student  s1 = new Student (111,"Karan");
    Student  s2 = new Student (222,"Aryan");
    s1.display();
    s2.display();
    }
}
```

Test it Now

Output111 Karan
       222 Aryan

id=222;

name=Aryan;

id=111;

name=Karan;

s2

s1

Stack Memory

Heap Memory
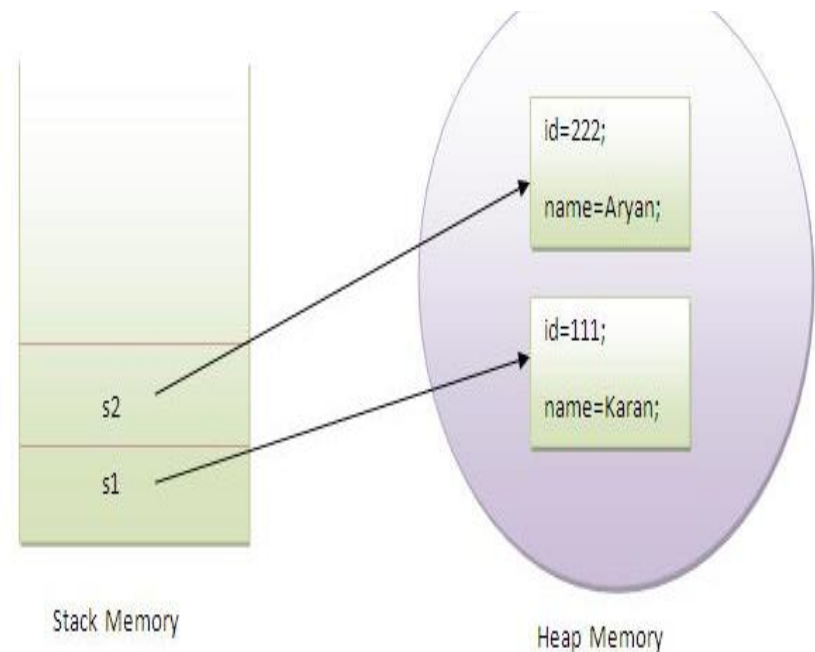
this − − 5 uses

2. this() :- invokes current class constructor.

```java
class Demo{
    Demo()
    {
        System.out.println("In first constructor");

    }
    Demo(String name){
        this(); //should be first line
        System.out.println("In Second constructor "+name);
    }

    public static void main(String args[]){

        Demo d= new Demo("John");

    }
}
```

this — 5 uses

3. this.method ():invoke current class method

```java
class Student {

    void one(){
        System.out.println("one" );
        this.two();
    }

    void two(){

        System.out.println("Wellcome to Wase" );
    }

    public static void main(String args[]){
        Student s = new Student();
        s.one();
    }
}
```

this – 5 uses

4. **Method(this):** to pass as an argument in the method call.

```
class Student {

    void two ( Student stud){
        System.out.println("two" );
        stud.three();
    }
    void three()
    {
        System.out.println("three");
    }
    void one()
    {
        System.out.println("one" );
        two(this);
    }

    public static void main(String args[]){
        Student s = new Student();
        s.one();
    }
}
```

this – keyword

5. return this : to return the current class instance.

Example: Rectangle .java

Syntax of class declaration

```
[ public ] [ ( abstract | final ) ]  class  ClassName

 {

        // variables

        // methods

 }
```

➢ **Modifiers** **and** **meaning**

Class type	Purpose
Public	can be seen by class of other package.

- A class can be
  - ➤ **public** class or
  - ➤ **default** (no modifier) class

---

public class Employee {
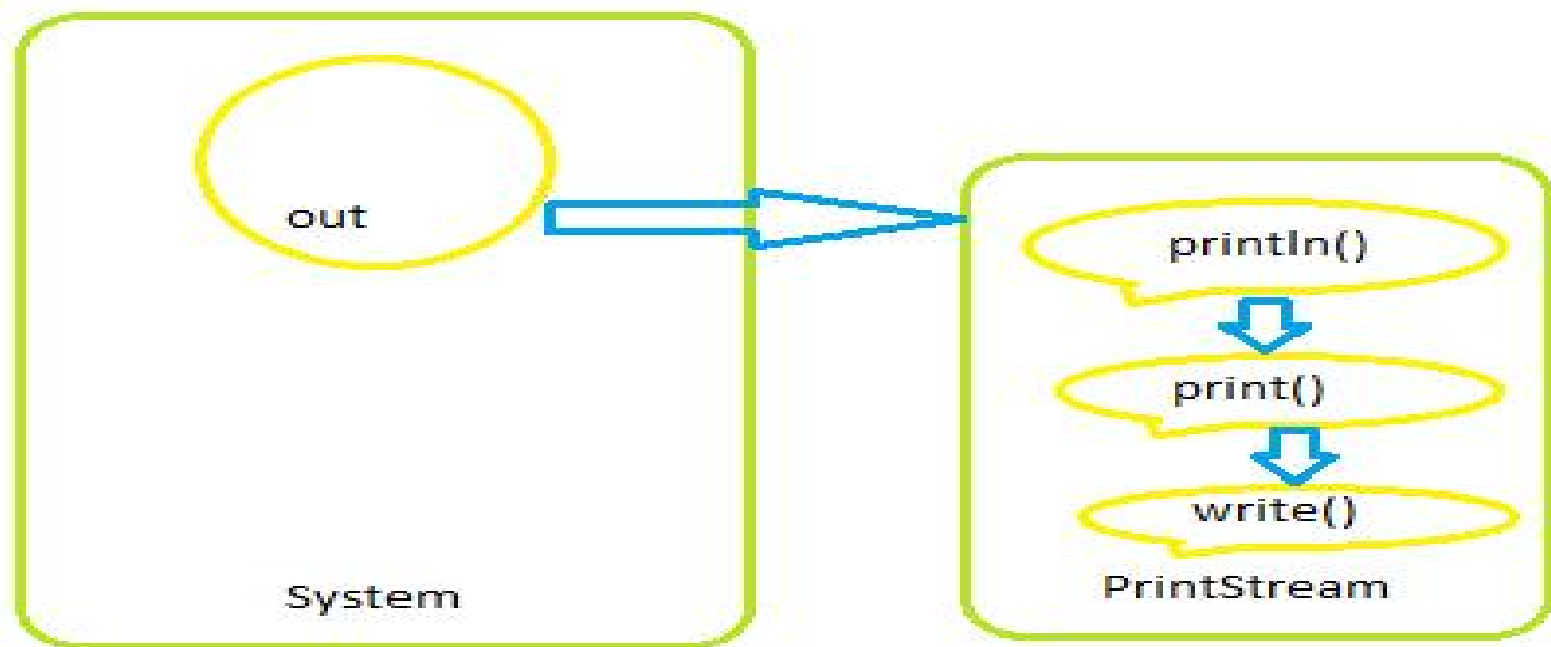
    ... ...

}

public class

---

class Employee {

    ... ...

}

default class

❑ There can be only one public class per source file.

❑ If there is a public class in a file,
   ➢ the name of the file must match the name of the public class.

❑ A file can have more than one nonpublic class.

❏ What is

  ➢ *System.out.println ( ).*

❑ What is

➢ *System.out.println ( ).*

❑ System

➢ *is a final class and cannot be inherited.*

❑ out
➢ *is a static member field.*
➢ *is public static object of PrintStream class.*
➢ *gets mapped with standard output and is ready to accept data.*

❑ println()
➢ *is a method of PrintStream class.*

- **Method overloading**

- *Constructor overloading*

Constructor and Method overloading

```java
class  Box  {

    double length, bredth, height;

    Box()   {         //default constructor

        length = bredth = height = 0;
    }

    Box( double len, double br, double ht )  {    //parameterized constructor

        length = len; bredth = br; height = ht;
    }

    void volume( double side )  {

        System.out.print ( " Volume is = " + side * side * side);
    }

    void volume()   {

        System.out.print ( "volume is = "+ length * bredth * height);
    }
}
```

Constructor overloading

Constructor and Method overloading

```java
class  Box  {

    double length, breadth, height;

    Box()   {         //default constructor

        length = breadth = height = 0;
    }

    Box( double len, double br, double ht )   {    //parameterized constructor

        length = len; breadth = br; height = ht;
    }

    void volume( double side )   {

        System.out.println ( " Volume is = " + side * side * side);
    }

    void volume()    {

        System.out.println ( "volume is = "+ length * breadth * height);
    }
}
```

Method Overloading

Constructor and Method overloading

```
class  Box Demo  {

    public static void main (String args []) {

        Box    b1 = new  Box (10, 20, 30);

        Box    b2 = new  Box ();

        b1.volume();

        b2.volume(30);
    }
}
```