15CSE202 Object Oriented Programming
Lecture 7

Class and Objects

Nalinadevi Kadiresan
CSE Dept.
Amrita School of Engg.

Syntax of Java Main Class

```
public class program-name {

    optional-variable-declarations-and-subroutines

    public static void main(String[] args) {
       statements
    }

    optional-variable-declarations-and-subroutines

}
```

Sample Hello World Program

```
/** A program to display the message
 *  "Hello World!" on standard output.
 */
public class HelloWorld {

   public static void main(String[] args) {
      System.out.println("Hello World!");
   }

}   // end of class HelloWorld
```

Writing to standard output

1. System.out.println()
2. System.out.print()
3. System.out.printf()

- Example:
  - System.out.println(“The value of x:”+ x)
  - System.out.print(“The value of x:”+ x)
  - System.out.println(x)
  - System.out.printf("%1.2f", amount);

Reading from standard input

- First, you should add the following line to your program at the beginning of the source code file, **before** the "public class...":

  ```
  import java.util.Scanner;
  ```

- Then include the following statement at the beginning of your main() routine:

  ```
  Scanner stdin = new Scanner( System.in );
  ```

- Reading from user using stdin
  - stdin.nextInt(), stdin.nextFloat(), stdin.nextDouble(), stdin.nextLong(), stdin.nextBoolean()

General Class structures in Java

```
Modifier  class Class-name
{
    variable declaration-1;
    variable declartion-2;
    method declaration-1;
    method declaration-2;
}
```

Variable declaration in a class

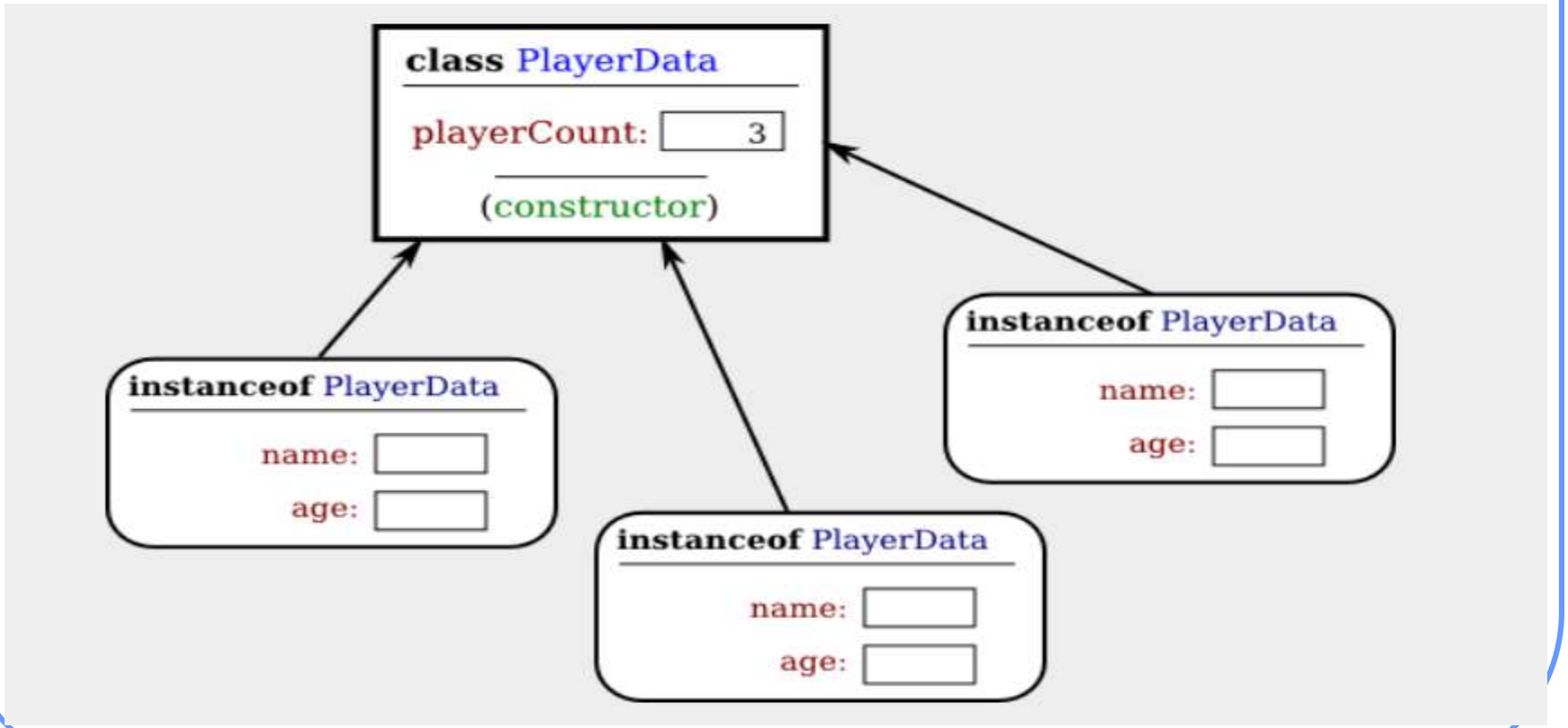
- **Modifier type-name variable-name-or-names**;
- Example:

```
class PlayerData
{
    public static int playerCount;
    public String name;
     int age;
}
```

Accessing variables

- the static variable playerCount is stored as part of the representation of the class in memory.
  - It is accessed as PlayerData.playerCount
  - There is only one of it
- the other two variables in the class definition are non-static
  - no such variable as ~~PlayerData.name~~ or ~~PlayerData.age~~
  - The variables are associated with its instance

Visualizing the PlayerData class



Method declaration in a class

```
modifiers  return-type  subroutine-name  ( parameter-list ) {
    statements
}
```

A simple Java Class – Student

```
public class Student {

   public String name;  // Student's name.
   public double test1, test2, test3;   // Grades on three tests.

   public double getAverage() {  // compute average test grade
      return (test1 + test2 + test3) / 3;
   }

} // end of class Student
```

Recap of Lifecycle of Objects

- Born healthy
  using constructors — **BRAHMA**
- Lives Safely
  using read/write functions — **VISHNU**
- Dies cleanly
  using destructors — **MAHESHWARA**

HOLY TRINITY of Object Oriented Programming

Now, the General Class structures in Java

```
Modifier  class Class-name
{
    variable declaration-1;     variable declaration-k;
   default constructor;
   parameterized constructor-1;
   parameterized constructor-n;

  getter /setter methods;

   protected void finalize( ) throws Throwable;

   method declaration-1;     method declaration-m;
}
```

BRAHMA

VISHNU

MAHESHWARA

Creating Objects in Java

In Java, no variable can ever hold an object.
A variable can only hold a reference to an object.

- **There is a special portion of memory called the heap where objects live.**
- **Instead of holding an object itself, a variable holds the information necessary to find the object in memory.**
- **This information is called a reference or pointer to the object.**

- declaring a variable does **not** create an object!

  Student std;

Creating Objects in Java Default Constructor

- only new operator creates a variable by calling default constructor

  std = new Student();

- The object itself is somewhere in the heap
- It is certainly **not at all true** to say that the object is "stored in the variable std."
- The proper terminology is that "the variable std refers to or points to the object,"

Creating Objects in Java Default constructor

- Facts,
  - The default constructor is invoked even if there is no constructor available in the class.
  - In such case, Java compiler provides a default constructor by default.

However, one can also write a default constructor in a class

Creating Objects in Java
Writing default constructor

Rules defined for the constructor.

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

Syntax of default constructor

modifier <class-name>() { <body> }

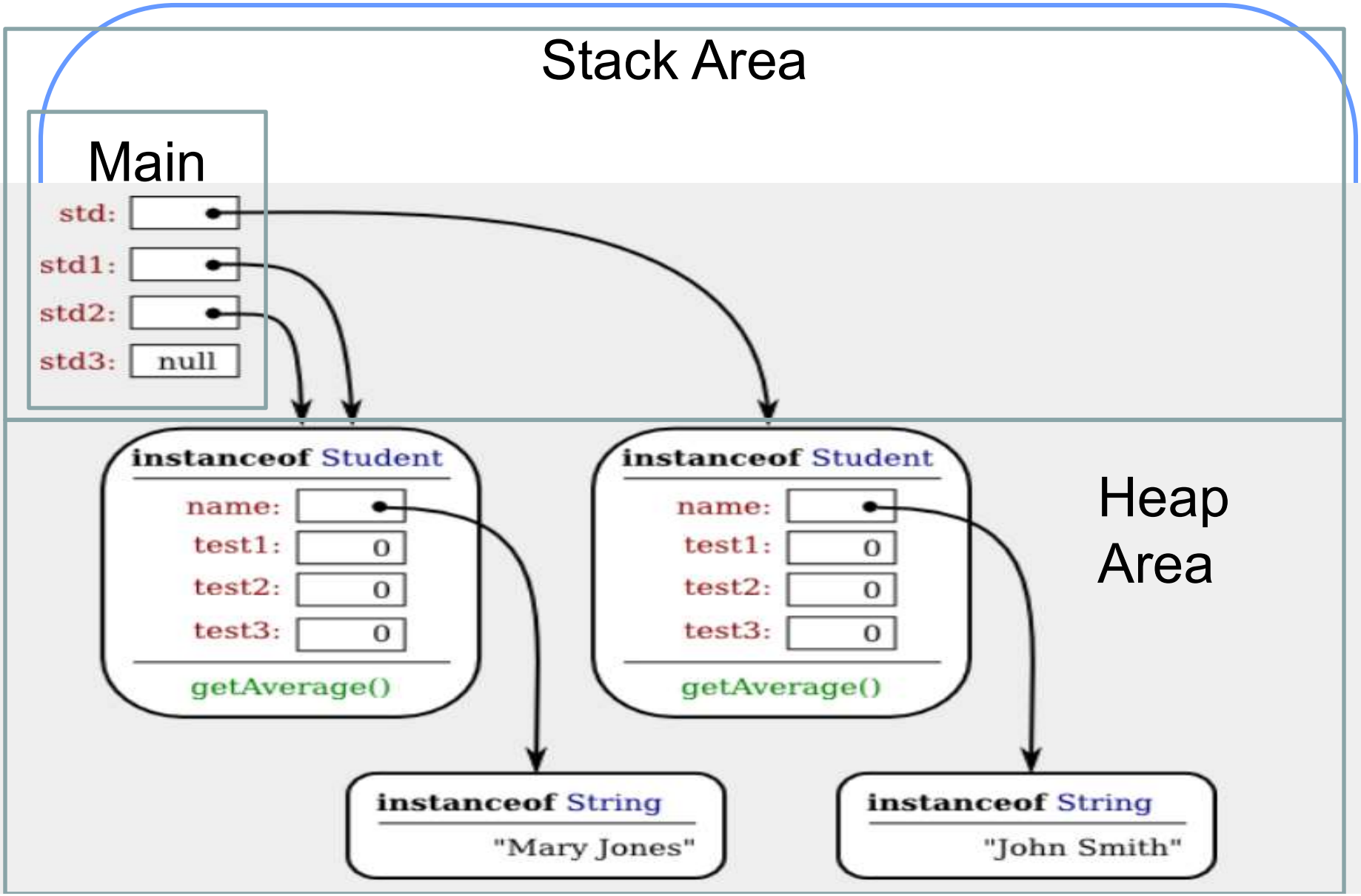
```
public class Student
 {
    public String name; // Student's name.
    public double test1, test2, test3; // Grades on three tests.
    public  Student()
    {
        System.out.println(" Creating Student !!! ");
     }
   public double getAverage()
   {   // compute average test grade
      return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```

Accessing member variables of an Object

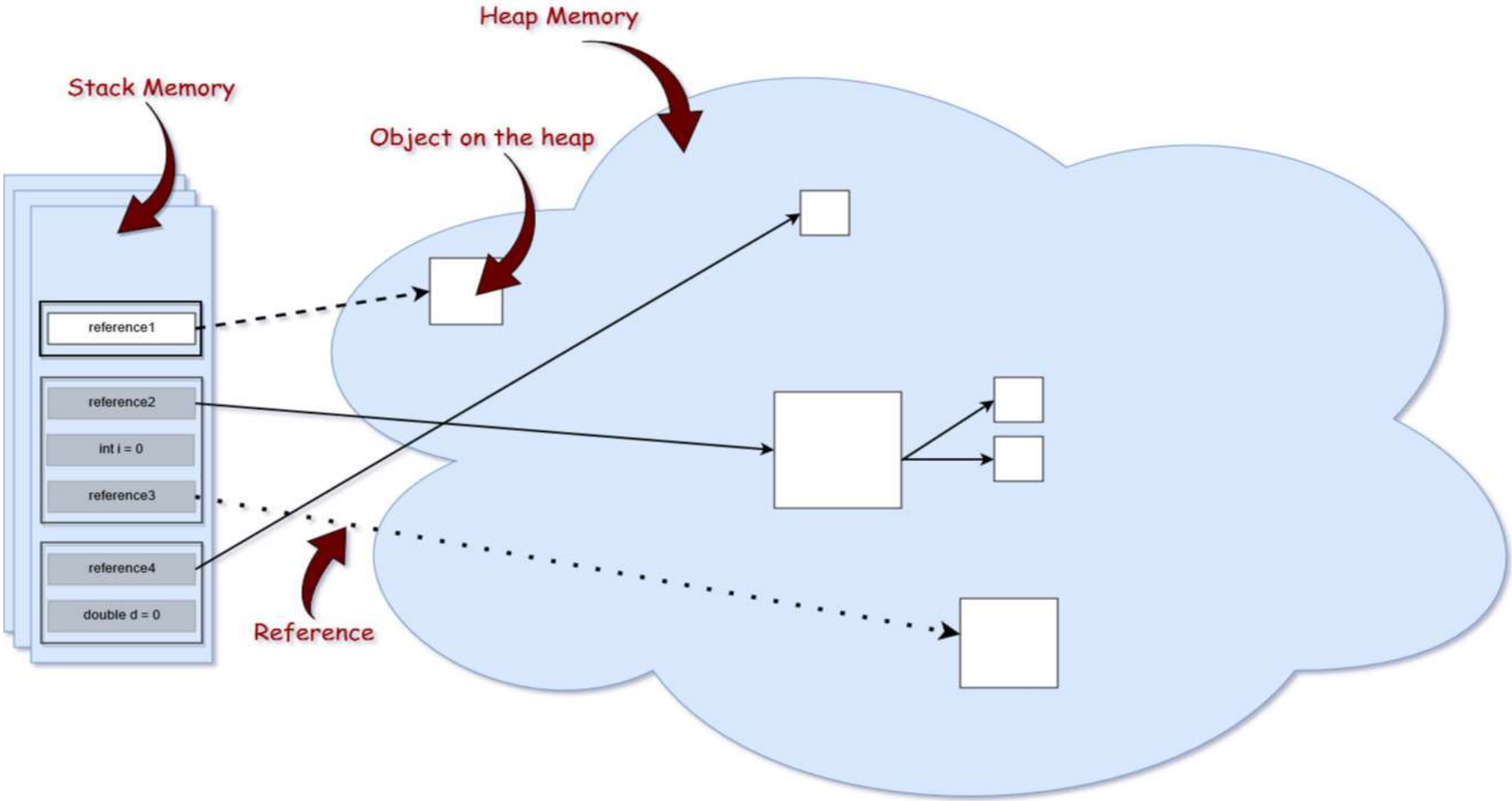
objectname. variablename

- suppose that the variable std refers to an object that is an instance of class Student.
- That object contains instance variables name, test1, test2, and test3.
- These instance variables can be referred to as std.name, std.test1, std.test2, and std.test3.

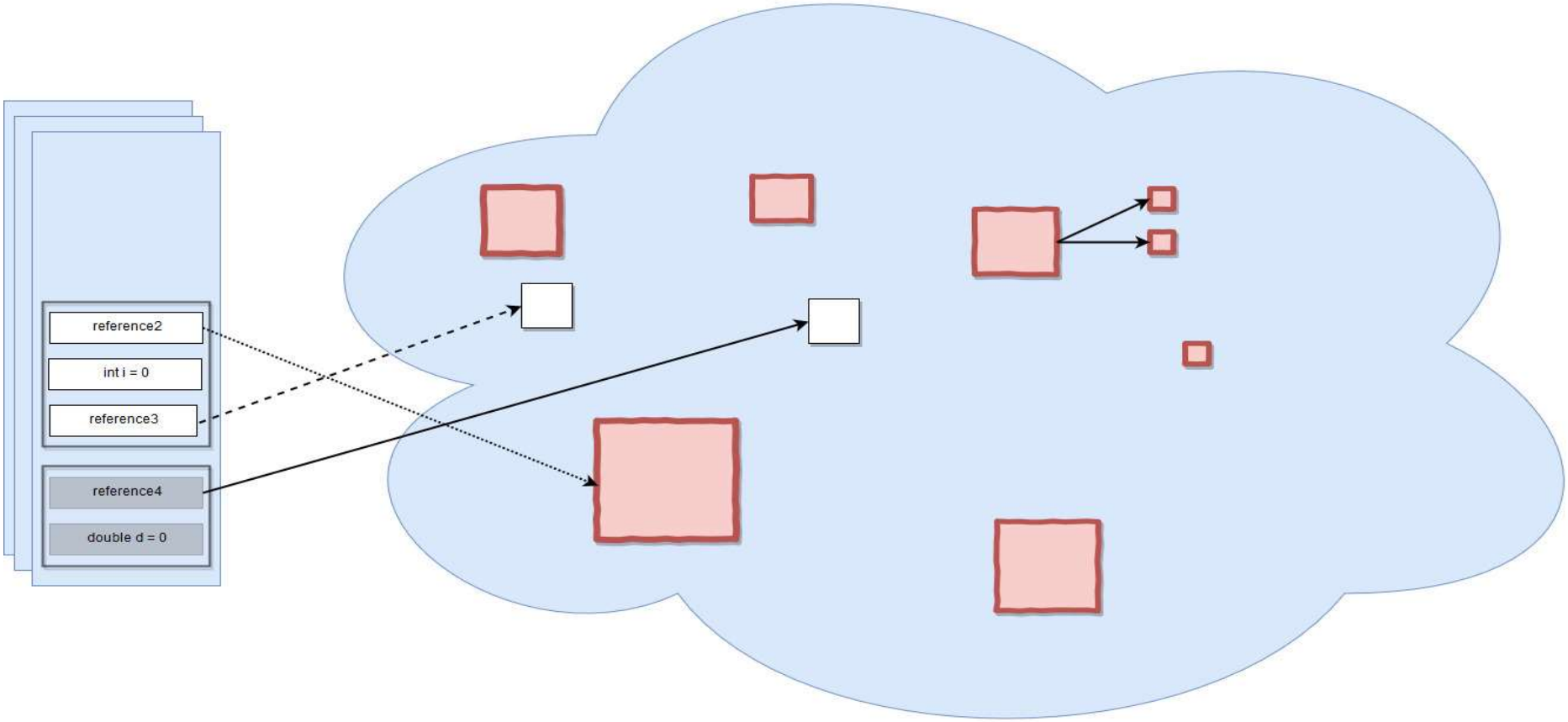
```
public class StudentDemo
{
    public static void main(String[] args)
    {
        Student std, std1,          // Declare four variables of
                  std2, std3;       //   type Student.

        std = new Student();        // Create a new object belonging
                                    //   to the class Student, and
                                    //   store a reference to that
                                    //   object in the variable std.

        std1 = new Student();       // Create a second Student object
                                    //   and store a reference to
                                    //   it in the variable std1.

        std2 = std1;                // Copy the reference value in std1
                                    //   into the variable std2.

        std3 = null;                // Store a null reference in the
                                    //   variable std3.

        std.name = "John Smith";  // Set values of some instance variables.
        std1.name = "Mary Jones";

            // (Other instance variables have default
            //    initial values of zero.)
```


Stack Area
Main
std:
std1:
std2:
std3:
null
instanceof Student
name:
test1:
0
test2:
0
test3:
0
getAverage()
instanceof Student
name:
test1:
0
test2:
0
test3:
0
getAverage()
Heap Area
instanceof String
"Mary Jones"
instanceof String
"John Smith"

General Memory view



Garbage eligible objects



Creating Objects in Jva

When one object variable is assigned to another, only a reference is copied. The object referred to is not copied.

The fact that variables hold references to objects, not objects themselves, has its consequence in defining variable constants which we shall learn it later.

Creating Objects in Java Parameterized constructors

- We can initialize the variables of the object at the time of creation

  Student std = new Student("Joe Martin");

- Syntax of parameterized constructor:

modifier <*class-name*>(*formal parameter-list*)

{

  <*body*>

}

```
public class Student
 {
    public String name; // Student's name.
    public double test1, test2, test3; // Grades on three tests.
    public  Student()
    {  System.out.println(" Creating Student !!! ");    }
    public  Student(String  s)
    {
        name= s;
     }
   public double getAverage()
   {   // compute average test grade
       return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```

Creating Objects in Java
Constructor Overloading

- Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
- They are arranged in a way that each constructor performs a different task.
- They are differentiated by the compiler by the number of parameters in the list and their types.

Example

```
Student std = new Student("Joe Martin");
Student s1 = new Student("Joe Martin",90,45, 54);
```

```
public class Student
 {
    public String name; // Student's name.
    public double test1, test2, test3; // Grades on three tests.
    public  Student()
    {  System.out.println(" Creating Student !!! ");    }
    public  Student(String  s)
    {    name= s;       }
    public  Student(String  s, int t1, int t2, int t3)
    {    name= s;
         test1=t1;
         test2=t2;
         test3=t3;
    }
} // end of class Student
```

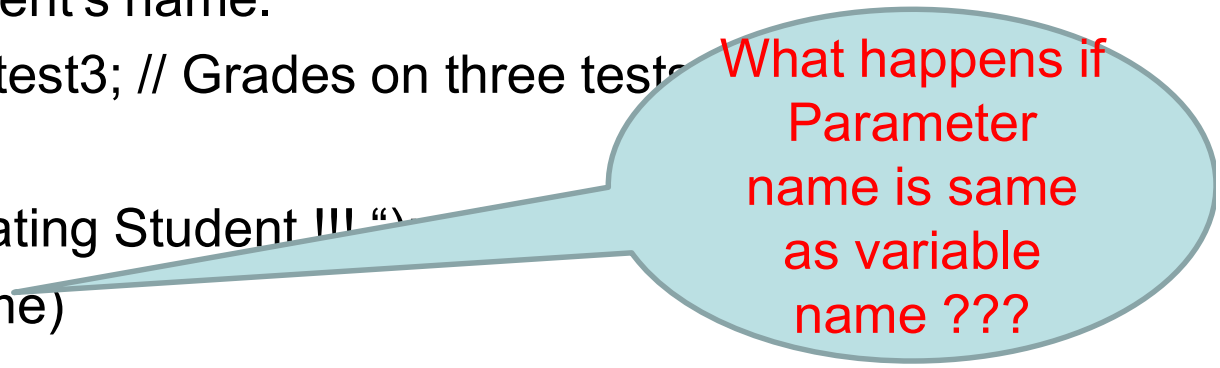
No copy constructor in Java.

However, objects can be parameters of constructor.

Difference between constructor and Java Methods

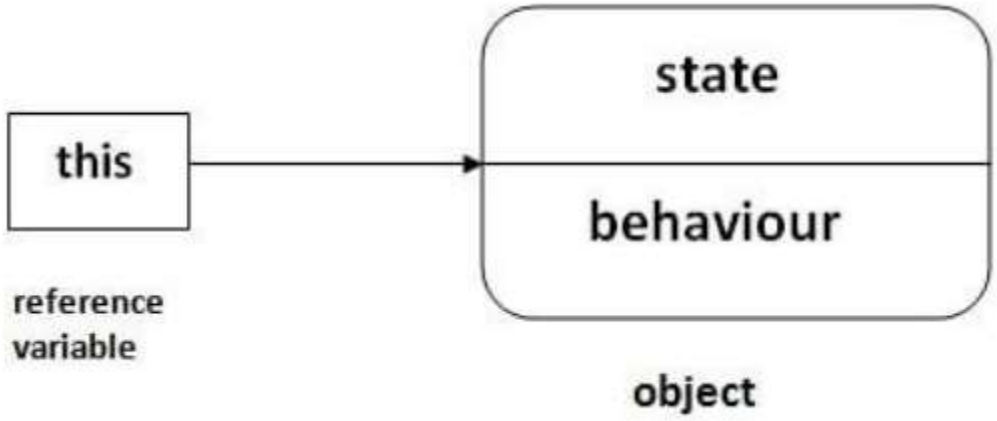
Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

```
public class Student
 {
    public String name; // Student's name.
    public double test1, test2, test3; // Grades on three tests
    public  Student()
    {  System.out.println(" Creating Student !!! ");
    public  Student(String  name)
    {
        name= name;
     }
     public  Student(String  name, int test1, int test2, int test3)
    {    name=name;      test1=test1;       test2=test2;         test3=test3;
       }
   public double getAverage()
   {   // compute average test grade
       return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```



Use this operator

- There can be a lot of usage of **java this keyword**.
- In java, this is a **reference variable** that refers to the current object.



```
public class Student
 {
    public String name; // Student's name.
    public double test1, test2, test3; // Grades on three tests
    public  Student()
    {  System.out.println(" Creating Student !!! ");
    public  Student(String  name)
    {
        this.name= name;
     }
     public  Student(String  name, int test1, int test2, int test3)
    {    this.name=name
         this.test1=test1;   this.test2=test2;   this.test3=test3;
      }
   public double getAverage()
   {   // compute average test grade
       return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```

Use this Operator

this operator

Here is given the 6 usage of java this keyword.

- this can be used to refer current class instance variable.
- this can be used to invoke current class method (implicitly)
- this() can be used to invoke current class constructor.
- this can be passed as an argument in the method call.
- this can be passed as argument in the constructor call.
- this can be used to return the current class instance from the method.

```
public class Student
{
   public String name; // Student's name.
   public double test1, test2, test3; // Grades on three tests.
   public  Student()
   {       System.out.println(" Creating Student !!! ");    }
   public  Student(String  name)
   {     this(); this.name= name;
    }
     public  Student(String  name, int test1, int test2, int test3)
   {    this(name);
        this.test1=test1;   this.test2=test2;   this.test3=test3;
     }
  public double getAverage()
  {   // compute average test grade
      return (test1 + test2 + test3) / 3;
   }
} // end of class Student
```

Note: Call to this must be first statement in constructor

How does this program work?

```
class S2{
  void m(S2 obj){
  System.out.println("method is invoked");
  }
  void p(){
  m(this);
  }
  public static void main(String args[]){
  S2 s1 = new S2();
  s1.p();
  }
}
```

How does this program work?

```
class A5{
void m(){
System.out.println(this);//prints same reference ID
}
public static void main(String args[]){
A5 obj=new A5();
System.out.println(obj);//prints the reference ID
obj.m();
}
}
```

Access Control of Objects

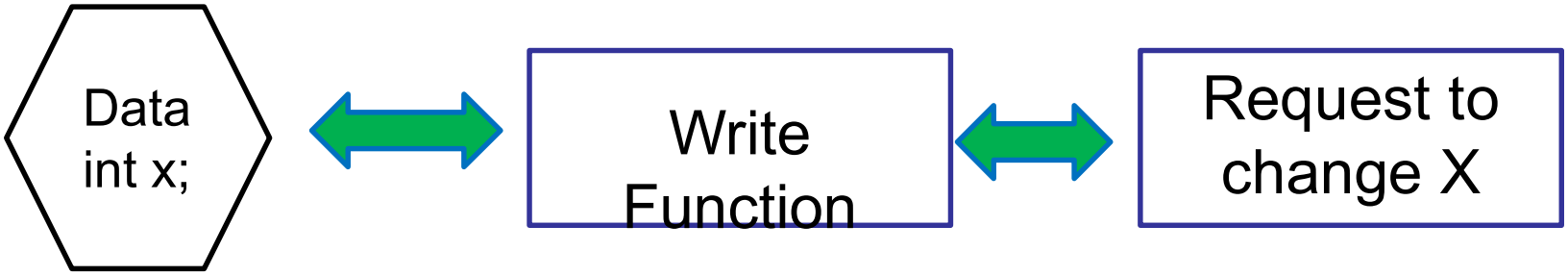
Getter and Setter methods

- Recall

member variables	Access Control
Public	makes it accessible from anywhere
Private	member can only be used in the class where it is defined

Write functions to Safeguard Data

$0 \leq x \leq 100$



Private!
Hidden !

Access Control of Objects

Getter and Setter methods

- So reading and writing private variables can be done using accessor and mutator methods
- Setter syntax:

```
public void set<Variablename>(Variabletype name)
{ <body>;        this. Variable-name =name;      }
```

- Getter syntax:

```
public return-type get<Variablename>()
{ <body>; return Variable-name; }
```

```
public class Student
 {
    private String name; // Student's name.
    private int age;  //Student age
    public Student(String name, int age)
    {    setName(name); setAge(age); }
    public void setName(String name) {  //setter method
           this.name=name;
     }
    public void setAge(int age) {   //setter method
         if (age < 110) { this.age=age; }
       else  { System.out.println(Abnormal age);
     }
     public  String getName()  {  return Name;}
     public int getAge()  { return age; }
} // end of class Student
```

Destructor in Java

- **Java lacks a destructor element, and instead uses a garbage collector for resource deallocation.**
- The finalize() method is inherited in all Java objects (since the Java convention is that everything is a sub-class of Object).
- This method is **NOT** a destructor!
- Instead, it is supposed to be used in order to provide **additional safety** in cases when you need to be sure that the use of external resources (like opening and closing a file, or a socket, or a stream) will be done correctly, which means closing everything before the program shutdown.

protected void finalize( ) throws Throwable { <body > }

Next Session will be

Static Keyword in Java