CST256: Object Oriented Programming

Kanak Kalyani

Syllabus

Unit I:

Features of Object Oriented Programming languages, Abstraction, Encapsulation, Inheritance, polymorphism and late binding. Concept of a class, Access control of members of a class, instantiating a class, constructor and method overloading.

Unit II:

Concept of inheritance, methods of derivation, use of super keyword and final keyword in inheritance, run time polymorphism, abstract classes and methods, Interface, implementation of interface, creating packages, importing packages, static and non-static members, Lambda Expressions Introduction, Block, Passing Lambda expression as Argument.

Unit III:

Exceptions, types of exception, use of try catch block, handling multiple exceptions, using finally, throw and throws clause, user defined exceptions, Introduction to streams, byte streams, character streams, file handling in Java, Serialization.

Syllabus

Unit IV:

Generics, generic class with two type parameter, bounded generics. Collection classes: Arraylist, LinkedList, Hashset, Treeset.

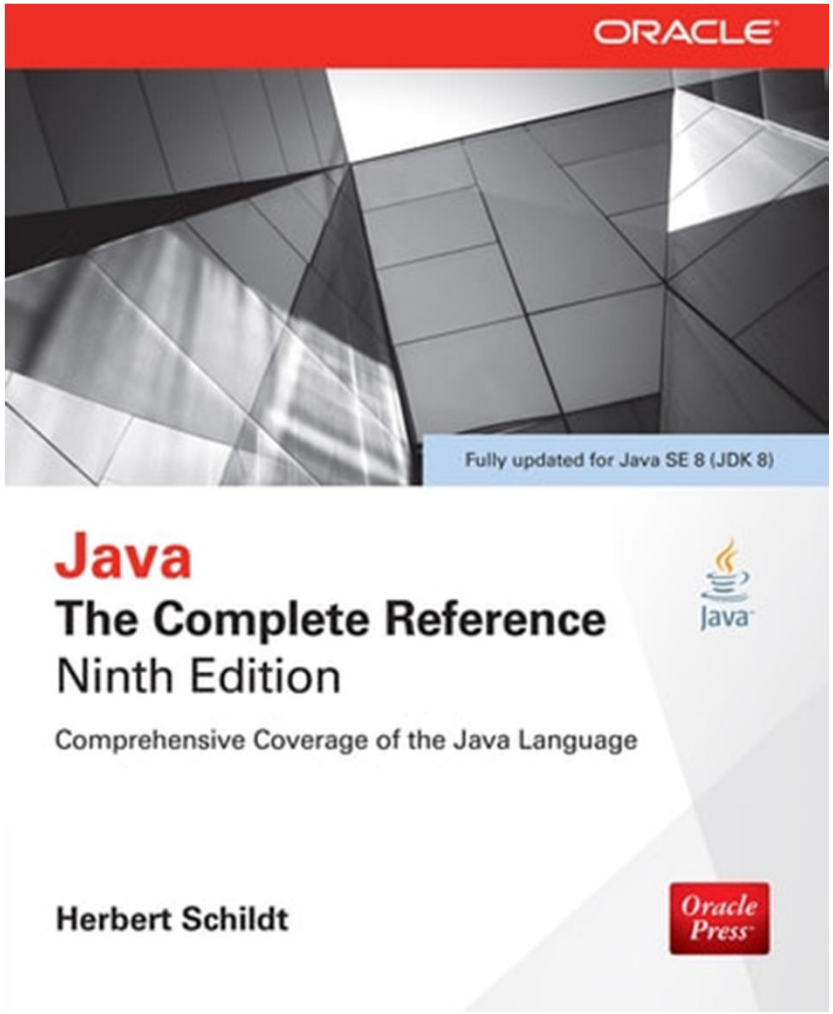
Unit V:

Multithreading: Java Thread models, creating thread using runnable interface and extending Thread, thread priorities, Thread Synchronization, InterThread communications.

Unit VI:

Introduction to Design Patterns, Need of Design Pattern, Classification of Design Patterns, Role of Design Pattern in Software design, Creational Patterns, Structural Design Patterns and Behavioral Patterns.

BOOK


ORACLE
Fully updated for Java SE 8 (JDK 8)
Java
The Complete Reference
Ninth Edition
Comprehensive Coverage of the Java Language
Herbert Schildt
Oracle Press

Course Outcomes (COs)

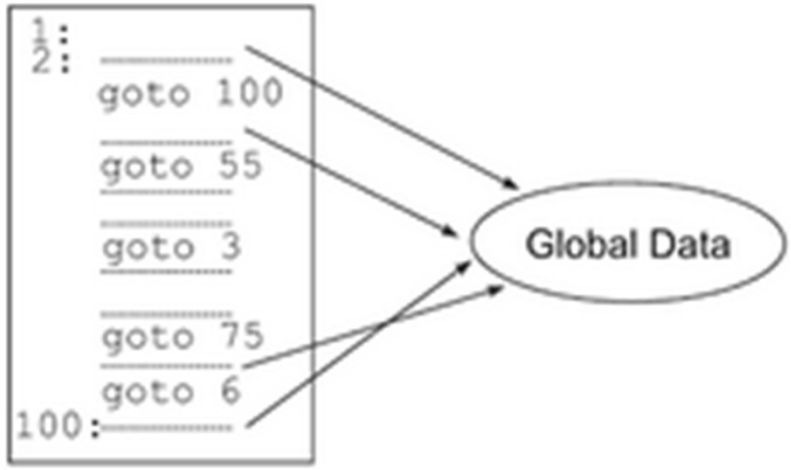
On successful completion of the course, students will be able to:

1. Understand the principles of object-oriented programming; create classes, instantiate objects and invoke methods.
2. Understand concept of generics and implement collection classes. Use exception handling mechanism.
3. Efficiently work with streams, use multithreading for solving classic synchronization problems. Perform java database connectivity and execute basic SQL commands.
4. Understand characteristics and need of Design Pattern in Software Design Process

Types of Programming Languages

Programming Paradigms

Monolithic Programming Paradigm

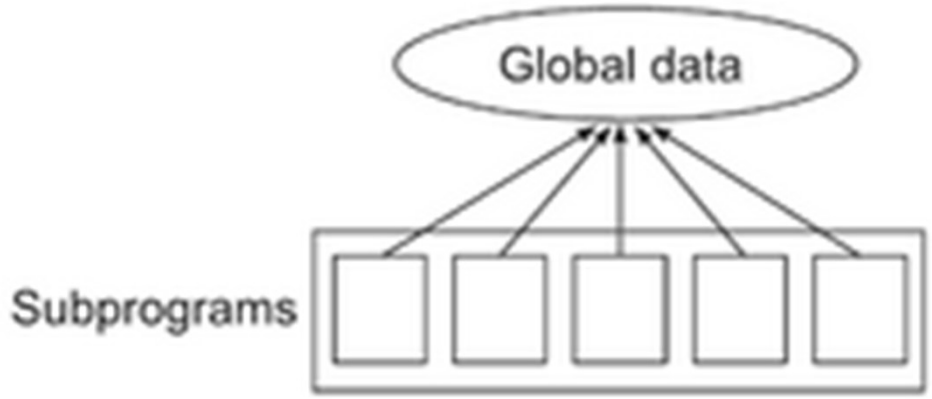


Example : Assembly Language and BASIC

Types of Programming Languages

Programming Paradigms

Procedural Programming

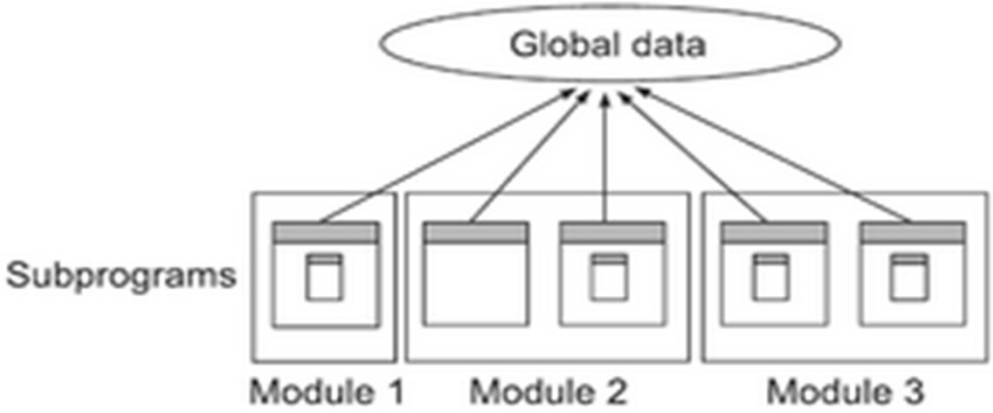


Example : COBOL and Fortran

Types of Programming Languages

Programming Paradigms

Structural Programming

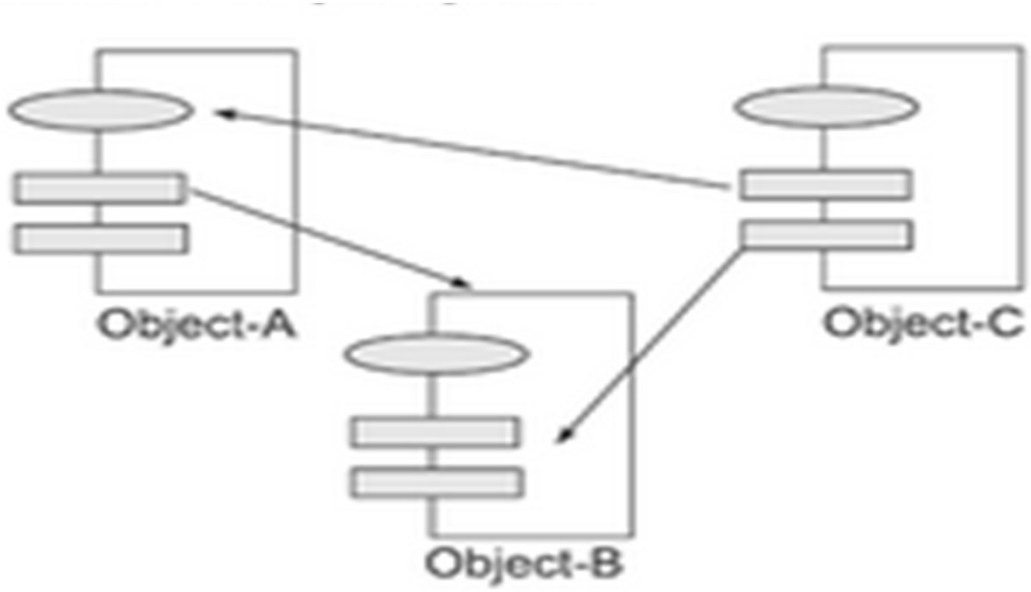


Example : C and Pascal

Types of Programming Languages

Programming Paradigms

Object Oriented Programming



Example : Java, C++, Python

FEATURES OF OBJECT-ORIENTED PROGRAMMING

- **Encapsulation**
- **Data Abstraction**
- **Inheritance**
- **Multiple Inheritance**
- **Polymorphism**
- **Delegation**
- **Genericity**
- **Persistence**
- **Concurrency**

Java Program

```
class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
        System.out.println("This is a simple Java program.");
        }
}
```

Java Program

```
class Example2 {
public static void main(String args []) {
        int num; // this declares a variable called num
        num = 100; // this assigns num the value 100
        System.out.println("This is num: " + num);
        num = num * 2;
        System.out.print("The value of num * 2 is ");
        System.out.println(num);
        }
}
```

This is num: 100
The value od num* 2 is 200

The Primitive Data Types

- **Integers** This group includes **byte**, **short**, **int**, and **long**, which are for whole-valued signed numbers.
- **Floating-point numbers** This group includes **float** and **double**, which represent numbers with fractional precision.
- **Characters** This group includes **char**, which represents symbols in a character set, like letters and numbers.
- **Boolean** This group includes **boolean**, which is a special type for representing true/false values.

Operators

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Relational Operators

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Boolean Logical Operators

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR (exclusive OR)
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

Bitwise Operators

- All of the integer types (except **char**) are signed integers.
- Negative Numbers are represented as 2's Complement.
- Example: 42 ➔ 00101010
  -42➔ 11010110

Bitwise Left Shift Operator

Example:-

int a = 64;

int b = a<<2;

b will have value 256

The Bitwise Operators

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

The Precedence of the Java Operators

Highest						
++ (postfix)	– – (postfix)					
++ (prefix)	– – (prefix)	~	!	+ (unary)	– (unary)	(*type-cast*)
*	/	%				
+	–					
>>	>>>	<<				
>	>=	<	<=	instanceof		
==	!=					
&						
^						
|						
&&						
||						
?:						
–>						
=	op=					
Lowest						

Literals

Commonly Used Literals

100 ➔ Integer Literal

98.6 ➔ Floating Point Literal

'X' ➔ Character Literal

"This is a test" ➔ String Literal

Integer Literal

- Any whole number value is an integer literal.
- Examples are 1, 2, 3, and 42.
- These are all decimal values, meaning they are describing a base 10 number.
- Other 2 bases supported are
  - Octal :- Octal values are denoted in Java by a leading zero
    - Example: 07
    - 09 will produce an error as 9 is not octal
  - Hexadecimal:- hexadecimal constant with a leading zero-x, (**0x** or **0X**)
    - Example:- 0X7f

Integer Literals

- Binary Value converted to integer

```
int x = 0b1010;
```

- Integer Literals for ease of Reading

```
int x = 123_456_789;
int x = 0b1101_0101_0001_1010;
```

Floating-Point Literals

- Standard Floating Point Literals

  2.0, 3.14159, 0.6667

- Exponential Floating Point Literals

  6.022E23, 314159E–05, 2e+100

- Floating-point literals in Java default to **double** precision. To specify a **float** literal, you must append **an *F* or *f*** to the constant

- Ease of Reading

  double num = 9_423_497_862.0;

  double num = 9_423_497.1_0_9;

Character Literals

Example: '*a*', '*z*', and '@'.

Escape Sequence	Description
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal Unicode character (xxxx)
\'	Single quote
\"	Double quote
\\	Backslash
\r	Carriage return
\n	New line (also known as line feed)
\f	Form feed
\t	Tab
\b	Backspace

For example, ' *141*' is the letter '*a*'.
For hexadecimal, enter a backslash-u (**\u**), then exactly four hexadecimal digits. For example, ' *u0061*' is the ISO-Latin-1 '*a*' because the top byte is zero.

Control Statements

The if Statement

```
if(num < 100)
        System.out.println("num is less than 100");
```

Operators supported

- < Less than
- > Greater than
- == Equal to

Loops

- **for Loop**

for*(initialization; condition; iteration) statement;*

```
class ForTest {
        public static void main(String args[]) {
        int x;
        for(x = 0; x<10; x = x+1)
        System.out.println("This is x: " + x);
        }
}
```

Loops

- **for each Loop**

for(*type itr-var* : *collection*) *statement-block*

```
class ForTest {
        public static void main(String args[]) {
                int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
                int sum = 0;
                /*for(int i=0; i < 10; i++){
                        sum += nums[i];
                }*/
                for(int x: nums){
                        sum += x;
                }
    }
}
```

Loops

while Loop

while(condition) statement;

```
class whileTest {
        public static void main(String args[]) {
        int x=0;
        while( x<10)
        System.out.println("This is x: " + x);
         x = x+1;
        }
}
```

The Scope and Lifetime of Variables

```
class Scope {
public static void main(String args[]) {
    int x; // known to all code within main
    x = 10;
    if(x == 10) { // start new scope
        int y = 20; // known only to this block
        // x and y both known here.
        System.out.println("x and y: " + x + " " + y);
        x = y * 2;
        }
    // y = 100; // Error! y not known here
    // x is still known here.
    System.out.println("x is " + x);
}
}
```

Type Conversion and Casting

Java's Automatic Conversions

When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

Example:

```
int a=10;
long l=a;
```

Type Conversion and Casting

Casting Incompatible Types

A *cast* is simply an explicit type conversion. It has this general form:

(*target-type*) *value*

Example 1:-

```
int i;
double d = 323.142;
i=(int) d;
```

Type Conversion and Casting

- **Automatic Type Promotion in Expressions**

```
class Promote {
public static void main(String args[]) {
    byte b = 42;
    char c = 'a';
    short s = 1024;
    int i = 50000;
    float f = 5.67f;
    double d = .1234;
    double result = (f * b) + (i / c) - (d * s);
    System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
    System.out.println("result = " + result);
    }
}
```

Arrays

One-Dimensional Arrays

type var-name[ ];

Example:

```
int month_days[];
month_days = new int[12];
month_days[1] = 28;
```

```
class AutoArray {
public static void main(String args[]) {
        int month_days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        System.out.println("April has " + month_days[3] + " days.");
}
}
```

Multidimensional Arrays

Example: int twoD[][] = new int[4][5];

```
class TwoDArray {
public static void main(String args[]) {
int twoD[][]= new int[4][5];
    int i, j, k = 0;
    for(i=0; i<4; i++){
        for(j=0; j<5; j++) {
                twoD[i][j] = k;
                k++;
        }
    }
    for(i=0; i<4; i++) {
        for(j=0; j<5; j++)
            System.out.print(twoD[i][j] + " ");
        System.out.println();
    }
  }
}
```

Alternative Array Declaration Syntax

type[ ] *var-name*;

int al[] = new int[3];

int[] a2 = new int[3];

char twod1[][] = new char[3][4];

char[][] twod2 = new char[3][4];

int nums[], nums2[], nums3[];

int[] nums, nums2, nums3;

Class

- Class is the core of Java.
- It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object
- Syntax:-

```
class classname
{
        type instance-variables; ...
        type methodname1(parameter-list) {
                // body of method
        }
}
```

Consider an Example

Functionalities
1)Lamp can be turned On
2) Lamp can be turned Off

Functionalities in class becomes METHODS (or functions)

to implement methods we need to store data in such a manner so that it is available to methods

For this particular case data is whether the lamp is on or off

Variable is:
boolean **isON**;

Methods are:
public void turnOn()
public void turnOff()

Class Lamp Looks Like

```
class Lamp
{
        // instance variable
            private boolean isOn;
        // method
            public void turnOn()
                    {
                            isOn = true;
                     }
         // method
             public void turnOff()
                    {
                        isOn = false;
                    }
}
```

OBJECTS

- An object is called an instance of a class.
- Syntax:

```
className object = new className();

// l1 object
Lamp l1 = new Lamp();
// l2 object
Lamp l2 = new Lamp();
```

What happens in MEMORY when object is created?

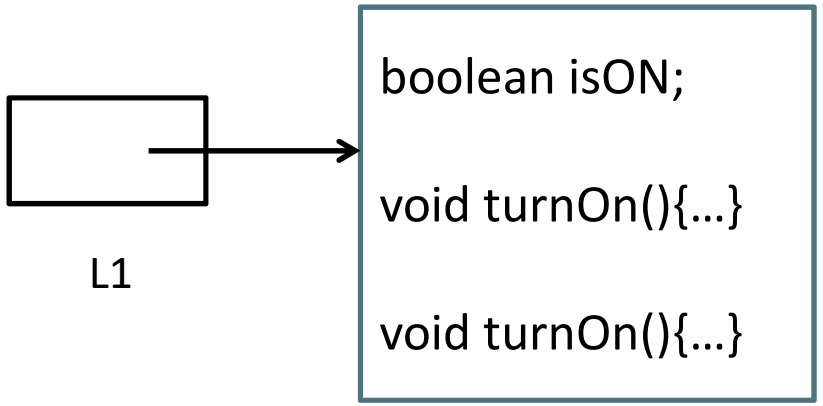
Statement

Lamp L1;

Effect

L1

L1 = new Lamp();



Final Program

```
class Lamp {
        boolean isOn;
        void turnOn(){
            // initialize variable with value true
            isOn = true;
            System.out.println("Light on? " + isOn);
        }
        void turnOff() {
            // initialize variable with value false
            isOn = false;
            System.out.println("Light on? " + isOn);
        }
}
```

Final Program

```
class Main
{
        public static void main(String[] args) {
                // create objects l1 and l2
                Lamp l1 = new Lamp();
                Lamp l2 = new Lamp();
                // call methods turnOn() and turnOff()
                 l1.turnOn();
                l2.turnOff();
                }
}
```

OUTPUT::
Light on? true
Light on? false

What is a class?

- A class is a template or blueprint from which objects are created.
- A class is a group of Objects, which have common properties
- It a logical entity

A Class contains

- ✓ Data Members(variables)
- ✓ Methods
- ✓ Constructors
- ✓ Blocks
- ✓ Nested Class(es) and Interface(s)

CONSTRUCTORS

- In Java, every class has its constructor that is invoked automatically when an object of the class is created. A constructor is similar to a method but in actual, it is not a method.

```
class Test {
        Test()
        {
                // constructor body
        }
}
```

Constructor name same as CLASSNAME

No RETURN TYPE of Constructor

Purpose of Constructor: To Initialize class members.

Lamp Example

```
class Lamp
{
        // instance variable
            private boolean isOn;

        //CONSTRUCTOR
        Lamp()
                {
                        isOn=False;
                }

        // method
            public void turnOn() {
                        isOn = true;
                 }
         // method
             public void turnOff() {
                    isOn = false;
                }
}
```

Types of Constructor

In Java, constructors can be divided into 3 types:

1. Default Constructor
2. No-Argument Constructor
3. Parameterized Constructor

Default Constructor

- If you do not create any constructors, the Java compiler will automatically create a no-argument constructor during run-time.
- This constructor is known as the default constructor. The default constructor initializes any uninitialized instance variables with default values.
- Example

Type	Default Value
`boolean`	false
`byte`	0
`short`	0
`int`	0

Example: Default Constructor

```
class DefaultConstructor
{
    int a;
    boolean b;
     public static void main(String[] args) {
        // A default constructor is called
        DefaultConstructor obj = new DefaultConstructor();
        System.out.println("a = " + obj.a);
        System.out.println("b = " + obj.b);
    }
 }
```

Output:

```
a = 0
b = false
```

No Argument Constructor

- A Java constructor may or may not have any parameters (arguments).
- If a constructor does not accept any parameters, it is known as a no-arg constructor.

For example,

```
Lamp(){
        isOn=False;
}
```

Parameterized Constructor

- Similar to methods, we can pass parameters to a constructor.
- Such constructors are known as a parameterized constructor.

For example,

```
Lamp(boolean value){
                    isOn=value;
}
.
.
.
Lamp l1 = new Lamp(true);
```

Example

Create a class Circle which provide functionalities to calculate circumference and area.

Every circle will have Center Coordinates and radius.[Properties]

Methods will be :

- calculateCircumference()
- calculateArea()

Example

```
 6  package corejava;
 7
 8  public class Circle {
 9      double x, y;// coordinates for Center
10      double r;// radius
11      double calculateCircumference()
12      {
13          return 2*Math.PI*r;
14      }
15      double calculateArea()
16      {
17          return r *r * (22/7) ;
18      }
19  }
20
```

Example

```
6   package corejava;
7
8   public class demo {
9       public static void main(String[] args) {
10          Circle c1=new Circle();
11          c1.x=10;
12          c1.y=20;
13          c1.r=10;
14
15          Circle c2=new Circle();
16          c2.x=5;
17          c2.y=5;
18          c2.r=20;
19
20          double circumfernce=c1.calculateCircumference();
21          double area=c1.calculateArea();
22
23          System.out.println("Circumfernce = " + circumfernce);
24          System.out.println("Area = " + area);
25
26      }
27  }
```

```
Output - COREJAVA (run)
run:
Circumfernce = 62.83185307179586
Area = 314.2857142857143
```

Improving Code by Adding Constructor

```
6   package corejava;
7
8   public class Circle {
9       double x, y;// coordinates for Center
10      double r;// radius
11
12      public Circle(double x, double y, double r) {
13          this.x = x;
14          this.y = y;
15          this.r = r;
16      }
17
18      double calculateCircumference()
19      {
20          return 2*Math.PI*r;
21      }
22      double calculateArea()
23      {
24          return r *r * (22/7) ;
25      }
26  }
27
```

Improving Code by Adding Constructor

```
6   package corejava;
7
8   public class demo {
9       public static void main(String[] args) {
10          Circle c1=new Circle(10,20,10);
11
12          Circle c2=new Circle(5,5,20);
13
14          double circumfernce=c1.calculateCircumference();
15          double area=c1.calculateArea();
16
17          System.out.println("C1 Circumfernce  = " + circumfernce);
18          System.out.println("C1 Area = " + area);
19
20           System.out.println("C2 Circumfernce  = " + c2.calculateCircumference() );
21          System.out.println("C2 Area = " + c2.calculateArea());
22
23      }
24  }
25
```

Constructor Overloading

Concept of having one than one constructor in class is Constructor Overloading

```
 8    public class Circle {
 9        double x, y;// coordinates for Center
10        double r;// radius
11
12        public Circle() {
13            this.x=0;
14            this.y=0;
15            this.r=1;
16        }
17
18        public Circle(double x, double y, double r) {
19            this.x = x;
20            this.y = y;
21            this.r = r;
22        }
23
24        double calculateCircumference()
```

```
Circle c1=new Circle(10,20,10);
Circle c2=new Circle();
```

Another Class

```
12  public class Rectangle {
13      double height, breadth;
14
15      public Rectangle() {
16          this.height = 1;
17          this.breadth = 1;
18      }
19
20      public Rectangle(double height, double breadth) {
21          this.height = height;
22          this.breadth = breadth;
23      }
24
25      double calculatePerimeter()     {
26          return 2*(height+breadth);
27      }
28      double calculateArea() {
29          return height*breadth;
30      }
31  }
32
```

Using more than 1 class

```
 */
package corejava;

public class demo {
    public static void main(String[] args) {
        Circle c1=new Circle(10,20,10);

        Rectangle r1=new Rectangle(10, 10);

        System.out.println("Circumfernce of Circle = " + c1.calculateCircumference() );
        System.out.println("Area of Circle = " + c1.calculateArea());

        System.out.println("Perimeter of Rectangle = " +r1.calculatePerimeter());
        System.out.println("Area of Rectangle = " + r1.calculateArea());

    }
}
```

```
Output - COREJAVA (run)
run:
Circumfernce of Circle = 62.83185307179586
Area of Circle = 300.0
Perimeter of Rectangle = 40.0
Area of Rectangle = 100.0
BUILD SUCCESSFUL (total time: 1 second)
```

Method Overloading

Method Overloading is a feature that allows a class to have more than one method having the same name

Method should follow at least one of the following:

- Number of parameters should be different
- Data type of parameters should be different

Example

- int add(int a, int b)
- int add(int a, int b, int c)
- int add(float a, float b)

Example

- Sarah got confused to calculate volume of cylinder and cuboid. Write a Java application to help Sarah to do this.

Create a class called VolumeCalculator that has the following methods

1. double calculateVolume(double radius,double height)
   This method calculates the volume of the cylinder using the formula 3.14*radius*radius*height
2. double calculateVolume(int length,int breadth,int height)
   This method calculates the volume of the cuboid using the formula length*breadth*height
   Write a TestMain class to test the application.

Example

```
12    class VolumeCalculator{
13            double calculateVolume(double radius,double height)
14        {
15            // returns volume of CYLINDER
16            return Math.PI * radius*radius*height;
17        }
18        double calculateVolume(int length,int breadth,int height)
19        {
20            //returns volume of CUBOID
21            return length*breadth*height;
22        }
23     }
24    public class VolumeCalculatorDemo{
25
26        public static void main(String[] args) {
27            VolumeCalculator vc=new VolumeCalculator();
28            double cylinderVolume = vc.calculateVolume(10 , 5);
29            System.out.println("Volume of Cylinder = " + cylinderVolume);
30
31            double cuboidVolume = vc.calculateVolume(10,20,30);
32            System.out.println("Volume of Cuboid = " + cuboidVolume);
33        }
34    }
```

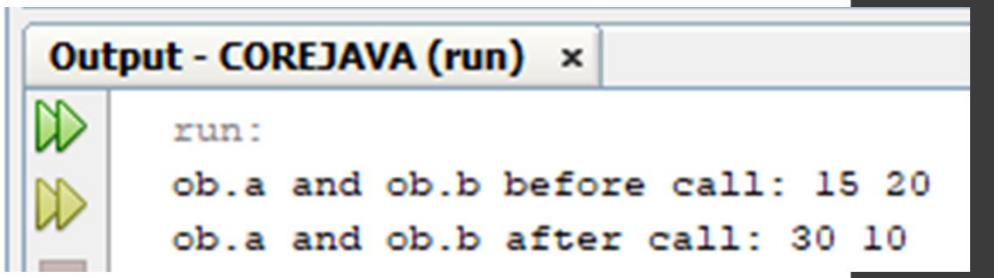
```
Output - COREJAVA (run)
run:
Volume of Cylinder = 1570.7963267948967
Volume of Cuboid = 6000.0
```

- Write a program to create a class Test. The class will have an overloaded method "Evaluate" which will decide whether the candidate is selected or not.
- For the post of Programmer, the candidate is evaluated on 4 parameters, Aptitude Test, Technical Test, Gradesheet Score and Personal Interview. The candidate should score atleast 80 Marks.
- For the post of Team Leader, the candidate is evaluated on the Technical Test and Personal Interviewparameters. The candidate should score atleast 85 Marks.
- For the post of Project Manager, the candidate undergoes only Personal Interviewand candidate should score atleast 90 Marks. Write main() to demonstrate the working of class Test.

Passing Object as Parameter

```
package corejava;
class Test {
    int a, b;
    Test(int i, int j) {
    a = i;
    b = j;
    }
    // pass an object
    void meth(Test o) {
    o.a *= 2;
    o.b /= 2;
    }
}
public class ObjectPassingDemo {
    public static void main(String args[]) {
        Test ob = new Test(15, 20);
        System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: " +  ob.a + " " + ob.b);

    }
}
```



Returning an Object

```
6   package corejava;
7   class Test1 {
8       int a;
9       Test1(int i) {
10          a = i;
11      }
12      Test1 incrByTen() {
13          Test1 temp = new Test1(a+10);
14          return temp;
15      }
16  }
17  public class ObjectReturnDemo {
18  public static void main(String args[]) {
19      Test1 ob1 = new Test1(2);
20      Test1 ob2;
21      ob2 = ob1.incrByTen();
22      System.out.println("ob1.a: " + ob1.a);
23      System.out.println("ob2.a: " + ob2.a);
24  }
25  }
```

```
Output - COREJAVA (run)
run:
ob1.a: 2
ob2.a: 12
```

Example

- Create a class Bank Account which stores basic information about customer like Name, Account Number, Balance.
- Provide Functionalities for withdraw, deposit and transfer money.

Example: Object as Parameter

```
public class BankAccount{
    private String Name;
    private int Acc_number;
    private double balance;

    public BankAccount(String Name, int Acc_number, double balance) {
        this.Name = Name;
        this.Acc_number = Acc_number;
        this.balance = balance;
    }
    void display()
    {
        System.out.println("Name:-"+Name);
        System.out.println("Account Number:-" + Acc_number);
        System.out.println("Balance:- "+ balance);
    }
```

```
28      public boolean withdraw(double amount)
29      {
30          if(balance>amount)
31          {
32              balance -= amount;
33               return true;
34          }
35          else
36          {
37              System.out.println("Insufficient Balance");
38              return false;
39          }
40
41      }
42      public boolean deposit(double amount)
43      {
44
45              balance += amount;
46              return true;
47      }
48      public boolean transfer(BankAccount B,double amount)
49      {
50          if(withdraw(amount))
51          {
52              B.deposit(amount);
53              return true;
54          }
55          else
56              return false;
57      }
58      public double getBalance() {
59          return balance;
60      }
```

```
8   public class BankAccountDemo {
9       public static void main(String[] args) {
10          BankAccount b1=new BankAccount("Ram", 123, 10000);
11          BankAccount b2=new BankAccount("Shyam", 456, 500);
12          b1.withdraw(1000);
13          b1.display();
14          b2.deposit(500);
15          b2.display();
16
17          //transfer 1000 from Rams' Account to Shyams' Account
18          boolean s=b1.transfer(b2, 1000);
19          if(s==true)
20          {
21              System.out.println("Transfer Successfull");
22              b1.display();
23              b2.display();
24          }
25          else
26          {
27              System.out.println("Transfer UnSuccessfull");
28              b1.display();
29              b2.display();
30          }
31      }
32  }
```

```
run:
Name:-Ram
Account Number:-123
Balance:- 9000.0
Name:-Shyam
Account Number:-456
Balance:- 1000.0
Transfer Successfull
Name:-Ram
Account Number:-123
Balance:- 8000.0
Name:-Shyam
Account Number:-456
Balance:- 2000.0
```

```
        boolean s=b1.transfer(b2, 10000);
19      if(s==true)
20      {
21          System.out.println("Transfer Successfull");
22          b1.display();
23          b2.display();
24      }
25      else
26      {
27          System.out.println("Transfer UnSuccessfull");
28          b1.display();
```

Output - COREJAVA (run)

```
Insufficient Balance
Transfer UnSuccessfull
Name:-Ram
Account Number:-123
Balance:- 9000.0
Name:-Shyam
Account Number:-456
Balance:- 1000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Example

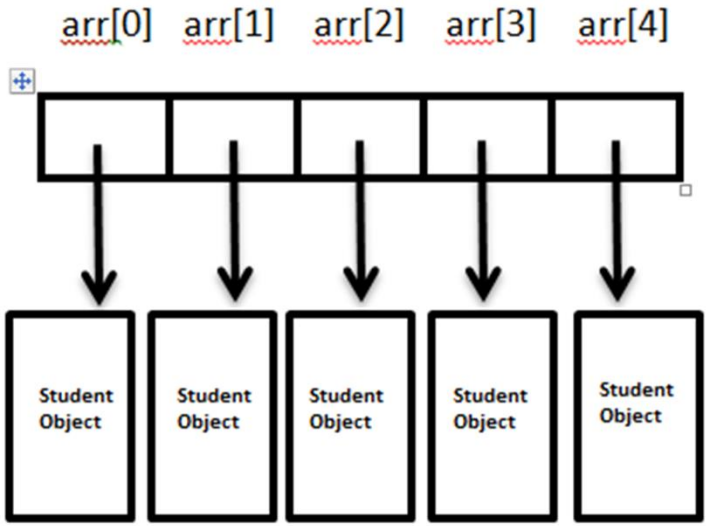
- Create a class Complex which stores real and imaginary part of complex number. Add functions in class complex to Add and Multiply two complex numbers. Write appropriate main().

Object Array: Example

```
14  class Student
15  {
16      public int roll_no;
17      public String name;
18      Student(int roll_no, String name)
19      {
20          this.roll_no = roll_no;
21          this.name = name;
22      }
23      void display()
24      {
25          System.out.println("Name: " + name + " Roll No. " + roll_no);
26      }
27  }
```

Object Array: Example

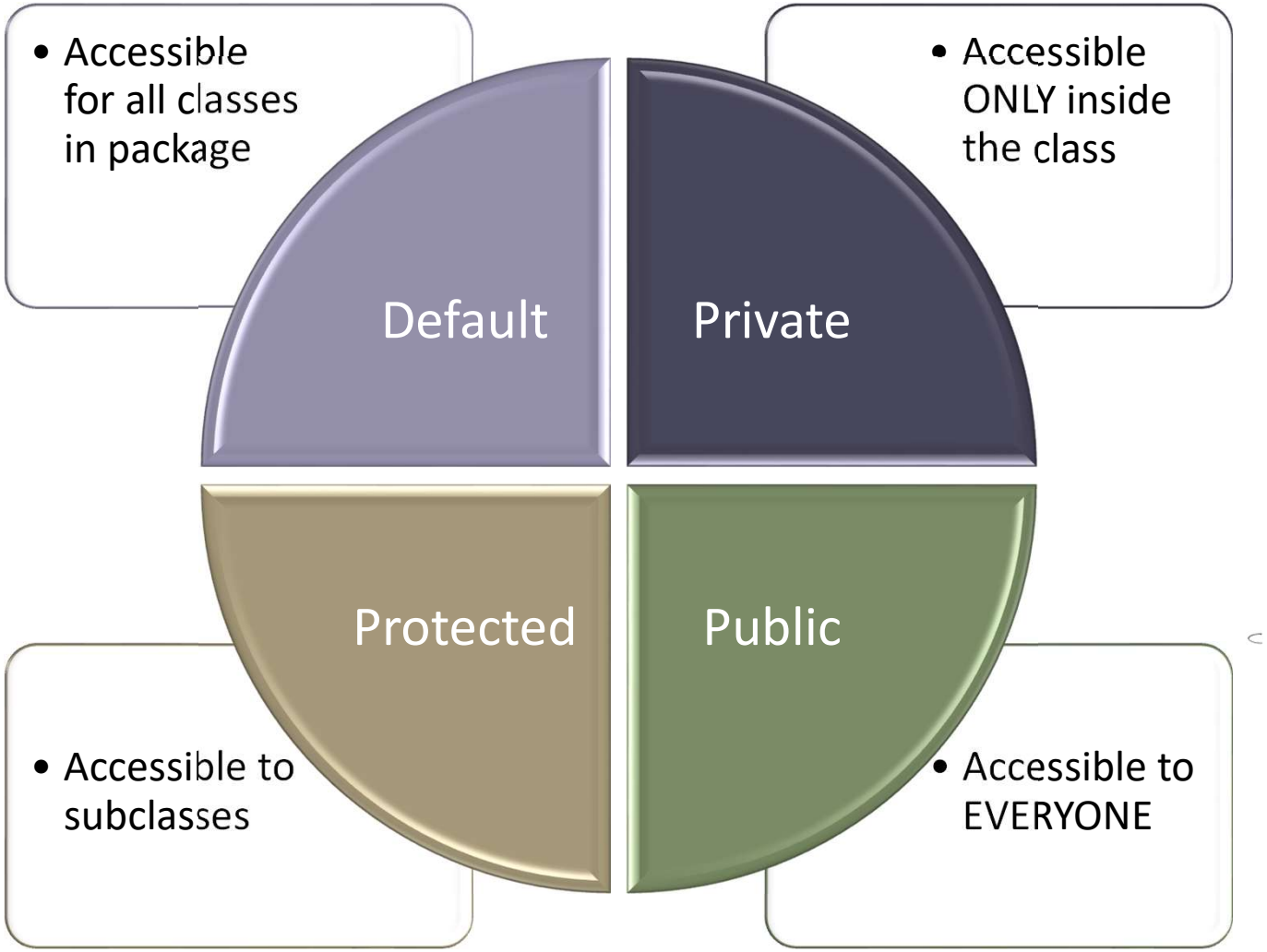
```
28  public class ObjectArrayDemo {
29      public static void main(String[] args)
30          Student[] arr;
31
32          arr = new Student[5];
33          // initialize the elements of the array
34          arr[0] = new Student(1,"aman");
35          arr[1] = new Student(2,"vaibha
36          arr[2] = new Student(3,"shikar
37          arr[3] = new Student(4,"dharme
38          arr[4] = new Student(5,"mohit"
39
40          // accessing the elements of t
41          for (Student arr1 : arr) {
42              arr1.display();
43          }
44      }
45  }
```



Object Array Example

- Create a class student which stores name, marks of 3 subjects and total marks of student. Create an array of student objects to store information about 5 students. Write a method to sort students on total marks. Write a method int Range(double, double) to count number of students in specified range.

Access Modifiers



Public Access Modifier

```
6   package P1;
7
8   public class A{
9       public void display()
10      {
11          System.out.println("This is Class A");
12      }
13  }
```

```
6   package P1;
7
8   public class Demo {
9       public static void main(String[] args) {
10          A ob=new A();
11          ob.display();
12      }
13  }
```

```
6   package P2;
7   import P1.A;
8
9   public class demo1 {
10      public static void main(String[] args) {
11          A ob=new A();
12          ob.display();
13      }
14  }
```

Private Access Modifier

```
6   package P1;
7
8   class A{
9       private void display()
10      {
11          System.out.println("This is Class A");
12      }
13  }
14
15  public class Demo {
16      public static void main(String[] args) {
17          A ob=new A();
            ob.display();
19      }
20  }
21
```

Default Access Modifier

```
6   package P1;
7
8   class A{
9       void display()
10      {
11          System.out.println("This is Class A");
12      }
13  }
14
15  public class Demo {
16      public static void main(String[] args) {
17          A ob=new A();
18          ob.display();
19      }
20  }
```

```
6   package P2;
7   import P1.A;
8
9   public class demo1 {
10      public static void main(String[] args) {
11          A ob=new A();
12          ob.display();
13      }
14  }
```

Protected Access Modifier

```
6   package P1;
7   public class A{
8       public void display()
9       {
10          System.out.println("This is Class A");
11      }
12      protected void display2()
13      {
14          System.out.println("This is Class A");
15      }
16  }
```

```
6   package P2;
7   import P1.A;
8   public class demo1 {
9       public static void main(String[] args) {
10          A ob=new A();
11          ob.display();
12          ob.display2();
13      }
14  }
```

```
6   package P2;
7   import P1.A;
8   public class demo1 extends A{
9       void show()
10      {
11          display();
12          display2();
13      }
14  }
```

Example

- Design a class Student for storing marks in 3 subjects.
- Provide functionality to calculate average and grade.
- Design a class student in such a way that direct access to data members is not possible.

Student Example

```
6   package corejava;
7   class student
8   {
9       private String Name;
10      private String Enroll;
11      private int[] marks;
12
13      public student(String Name, String Enroll, int[] marks) {
14          this.Name = Name;
15          this.Enroll = Enroll;
16          this.marks = marks;
17      }
18
19      public student() {
20          this.Name = "";
21          this.Enroll = "";
22          this.marks = null;
23      }
24      public void setData(String Name, String Enroll, int[] marks)
25      {
26          this.Name = Name;
27          this.Enroll = Enroll;
28          this.marks = marks;
29      }
```

Student Example

```
31      public String getName() {
32          return Name;
33      }
34      public int[] getMarks() {
35          return marks;
36      }
37      public String getEnroll() {
38          return Enroll;
39      }
40      public double calculateAverage()
41      {
42          int sum=0;
43          for(int m:marks)
44          {
45              sum = sum + m;
46          }
47          return (double)sum/marks.length;
48      }
```

Student Example

```
49      public String calculateGrade()
50      {
51          double average=calculateAverage();
52          if(average>=80)
53              return "O";
54          else if(average>=60)
55              return "A";
56          else if(average>=40)
57              return "B";
58          else
59              return "F";
60      }
61
62      public void display()
63      {
64          System.out.println("Name of Student: " + Name);
65          System.out.println("Enrollnment Number: " + Enroll);
66          System.out.println("Average Marks: " + calculateAverage());
67          System.out.println("Grade is: " + calculateGrade());
68      }
69  }
```

Student Example

```
70  public class StudentDemo {
71
72      public static void main(String[] args) {
73          student s1=new student();
74          int marks[]={50,60,70};
75          s1.setData("Amit", "BECSU015", marks);
76          s1.display();
77
78          int marksObject[]=s1.getMarks();
79          System.out.print("Marks are:-  " );
80          for(int m:marksObject)
81              System.out.print(m + " ");
82      }
83
84  }
85
```

```
Output - COREJAVA (run)
run:
Name of Student: Amit
Enrollnment Number: BECSU015
Average Marks: 60.0
Grade is: A
Marks are:-  50 60 70 BUILD SUCCESSFUL
```

Example

As the winter season is nearing ABC Air Coolers has decided to go for a clearance sale for 4 days.

The number of items and the price for each item is displayed on the board. The price of the unsold items will be reduced by 7% on each day.

For example if an item price is Rs.10000 on the first day,
then it will cost Rs.9300 for the next day. Again if the items are unsold on the third day then the price goes to Rs.8649 which is 7% of 9300.

Design a program that prompts the user for the name of the item, number of items for sale ,number of items sold on each day .
Display the name of the item, Price of the item on each day and the number of items available for sale.

Also display the Total amount obtained through the clearance sale.

What will be the output??

```
class A
{
    A()
    {
       // Constructor of Class A
    }
    A1()
    {



    }
}
```

Can a Constructor be declared as private??

```
class A
{
    private A()
    {
        // Private Constructor
    }

    void methodOne()
    {
        //You can use private constructor inside the class
        A a1 = new A();
    }
}

class MainClass
{
    public static void main(String[] args)
    {
        //You can't use private constructor outside the class like this
        // A a1 = new A();
    }
}
```

Recursive constructor calling is not allowed.

```
class A
{
    A()
    {
        this();
        // It gives compile time error
    }
}
```

No Cylic calling of constructors.

```
class A
{
    A()
    {
        this(10);
        // It gives compile time error
    }
    A(int i)
    {
        this();
        // It gives compile time error
    }
}
```

Practice Problems

- Write a program to create a class Matrix which will store a 2D matrix.
  - Add Functionalities to add, multiply 2 matrices and store the result in 3rd matrix.
  - Add the functionality to find transpose of matrix.
  - Display the matrix in proper format.
  - Write a menu driven program to perform the operations.
  - (Take input from user with the help of scanner class)

Practice Problems

- Write a class called Product. It should contain the following information: product code, name of product, cost of product, and quantity of product currently in stock. Assume code and name are represented by strings of characters. Include following constructor and methods in the class definition.
    1. A constructor Product(code,name) which creates a new product with the given code and name. Initially, the cost and the quantity of the product should be set to zero.
    2. An instance method getName() that will return the name of the product.
    3. An instance method addStock(int n) that will add n to the quantity of the product in stock.
    4. An instance method outOfStock() that will return the value true if there is none of this product in stock. Otherwise it will return false.
    5. Write an appropriate main to create array of objects of class product and use them.

Practice Problems

- Write a program to create a class Vector with data member as single dimensional array of integers to store the elements in vector. Include methods to insert value in array. Add a function to add two vectors and display the resultant vector. Add a display method to display the contents of the array. Create objects to demonstrate functionalities.