Object Oriented Programming with Java 18IS34

Object Oriented Programming with Java (Theory)

Course Code	18CS/IS34	**Credits**	03
Course type	PC	**CIE Marks**	50 marks
Hours/week: L-T-P	3 – 0 – 0	**SEE Marks**	50 marks
Total Hours:	Lecture = 40 Hrs; Tutorial = 00 Hrs Total = 40 Hrs	**SEE Duration**	3 Hours for 100 marks

Books	
	Text Books:
1.	Herbert Schildt& Dale Skrien, "Java Fundamentals A Comprehensive Introduction", TMH. Special Indian edition.
	Reference Books:
1.	1. Kathy Sierra & Bert Bates, "Head First Java", O'Reilly, 2nd Edition and onwards.

Introduction

- Java was conceived by James Gosling and others at Sunmicrosystems in 1991.
- Need for platform independent language.
- To create software to be embedded in various platform independent elctronic devices.
- Most of the languages were designed.
- C++ requires compiler that targets that CPU.
- Different computers have different types of CPU and OS, hence requires cross platform language.
- Internet has different types of computers.
- Portability is required for a language.

Java contribution to internet

- Java innovated applet.
- Downloaded code may harm the computer.
- Java execution environment.
- Portability: byte code.
- JVM(Java Virtual Machine)
- JIT(Just in time) Compiler
- JDK (Java Development Kit)
- javac:
  java:

Simple Java Program

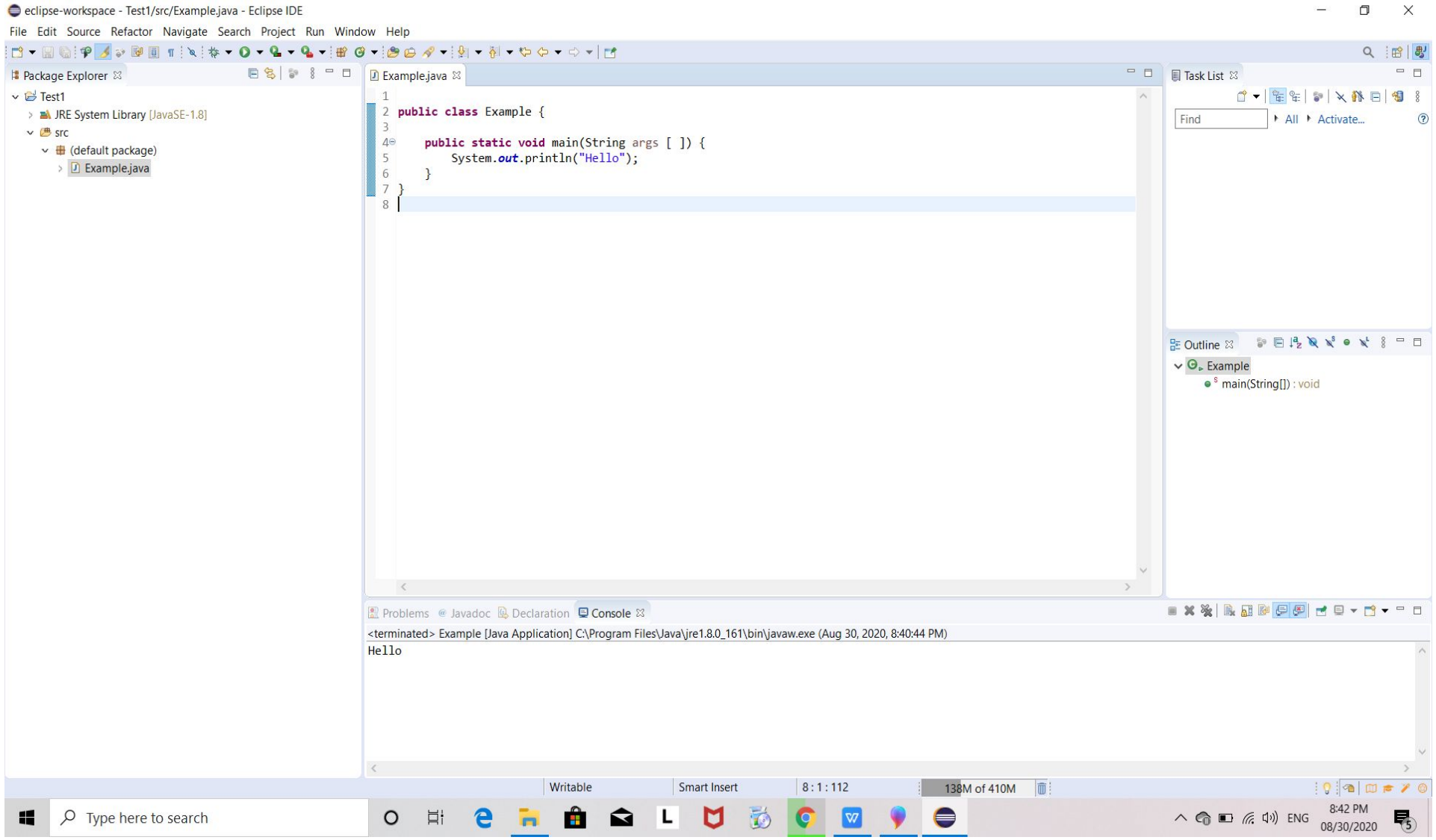
```
/* This is simple java program*/

class Example
{
    public static void main(String args [ ]) {
        System.out.println(“Java drives the Web”);
    }
}
```

Compiling the java program

javac Example.java

Example.class

java Example


eclipse-workspace - Test1/src/Example.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
Test1
JRE System Library [JavaSE-1.8]
src
(default package)
Example.java
Example.java
1
2 public class Example {
3
4 public static void main(String args [ ]) {
5 System.out.println("Hello");
6 }
7 }
8
Task List
Find
All
Activate...
Outline
Example
main(String[]) : void
Problems
Javadoc
Declaration
Console
<terminated> Example [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Aug 30, 2020, 8:40:44 PM)
Hello
Writable
Smart Insert
8 : 1 : 112
138M of 410M
Type here to search
ENG
8:42 PM
08/30/2020

Simple java program

Class Example

- class is java's basic unit of encapsulation.
- Example is name of the class
- { }
- Elements between braces are members of the class.

public static void main(String args [ ])

- public is an access modifier
- private is also an access specifier.

Key attributes of Object Oriented Programming Language

- encapsulation
- polymorphism
- inheritance

Second simple example

- Variable is a named memory location that can be assigned a value.

```
class Example2 {
        public static void main(String args [ ]){
                int var1;
                int var2;
                var1=1024;
                System.out.println(“var1 contains” + var1);
                var2=var1 / 2;
                System.out.println(“var2 contains var1/2=” + var2);
        }
}
```

O/P: var1 contains 1024

var2 contains var1/2 = 512

data types and control statements

- Java defines two floating data types float and double.

double x; //declares double variable

- Conditional statements
  - if-else
  - nested if
  - if else if ladder
  - switch
  - nested switch
- Looping statements
  - for
  - while
  - do while

if-else statement

```
if (condition)
    statement;
else
    statement;

var1=10;
var2=11;
if(var1 < var2)
    System.out.println(“var1 is less than var2\n”);
else
    System.out.println(“var2 is less than var1\n”);
```

nested-if

- A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement

```
if (condition1)
{
   // Executes when condition1 is true
   if (condition2)
   {
      // Executes when condition2 is true
   }
}
```

nested-if

```
// Java program to illustrate nested-if statement
class NestedIfDemo
{
   public static void main(String args[])
   {
      int i = 10;
       if (i == 10)
      {
         // First if statement
          if (i < 15)
                        System.out.println("i is smaller than 15");
                       // Nested - if statement  Will only be executed if statement above  it is true
          if (i < 12)
                        System.out.println("i is smaller than 12 too");
                else
                       System.out.println("i is greater than 15");
      }
   }
}
```

if-else-if ladder

- **Java if-else-if ladder is used to decide among multiple options.**

```
class GFG {
  public static void main(String[] args)
  {       // initializing expression
         int i = 20;
         if (i == 10)    // condition 1
                System.out.println("i is 10\n");
        else if (i == 15)    // condition 2
                        System.out.println("i is 15\n");
         else if (i == 20)   // condition 3
                  System.out.println("i is 20\n");
                else
                        System.out.println("i is not present\n");
                System.out.println("Outside if-else-if");
               }
}
```

O/P

1. Program starts.
2. i is initialized to 20.
3. condition 1 is checked. 20 == 10, yields false.
4. condition 2 is checked. 20 == 15, yields false.
5. condition 3 is checked. 20 == 20, yields true.
   5.a) "i is 20" gets printed.
6. "Outside if-else-if" gets printed.
7. Program ends.

switch statement

- Use the switch statement to select one of many code blocks to be executed.

```
public class Test {
        public static void main(String[] args)
        {       int day = 5;
                String dayString;
                switch (day) {  // switch statement with int data type
                case 1:             dayString = "Monday";           break;
                case 2:             dayString = "Tuesday";          break;
                case 3:             dayString = "Wednesday";        break;
                case 4:             dayString = "Thursday";         break;
                case 5:             dayString = "Friday";           break;
                case 6:             dayString = "Saturday";         break;
                case 7:             dayString = "Sunday";                   break;
                default:    dayString = "Invalid day";              break;
                }
                System.out.println(dayString);
        }
}
```

nested switch

```
public class Test {
	public static void main(String[] args)
	{	String Branch = "CCE";
		int year = 2;
		switch (year) {
		case 1:	System.out.println("elective courses : Advance english, Algebra");	break;
		case 2:	switch (Branch) // nested switch
			{	case "CSE":
				case "CCE":
					System.out.println("elective courses : Machine Learning, Big Data");	break;
				case "ECE":
					System.out.println("elective courses : Antenna Engineering");	break;
				default:
				System.out.println("Elective courses : Optimization");
			}
		}
	}
}
```

Looping statements

- **for loop: for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.**

```
class forLoopDemo
{
   public static void main(String args[ ])
   {
      // for loop begins when x=2
      // and runs till x <=4
      for (int x = 2; x <= 4; x++)
         System.out.println("Value of x:" + x);
   }
}
```

O/P: Value of x:2

Value of x:3

Value of x:4

While loop

- **while loop: A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement**

```
class whileLoopDemo
{
   public static void main(String args[])
   {
      int x = 1;
       while (x <= 4)     // Exit when x becomes greater than 4
      {
         System.out.println("Value of x:" + x);          // Increment the value of x for next iteration
         x++;
      }
   }
}
```

O/P: Value of x:1

Value of x:2

Value of x:3

Value of x:4

do-while

- **do while: do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.**

```
class dowhileloopDemo
{
   public static void main(String args[])
   {
      int x = 21;
      do
      {          // The line will be printed even  if the condition is false
         System.out.println("Value of x:" + x);
         x++;
      }
      while (x < 20);
   }
}
```

O/P: Value of x: 21

Array

- An array is collection of variables of same datatype referred to by a common name
- One dimensional array

```
type [ ] arrayname = new type[size];
```

```
int [ ] sample =  new int [10];
```

- same can be declared like this

```
int [ ]  sample;
sample= new int [10];
```

Array

This is sample[0]: 1

This is sample[1]: 2

.

.

.

.

.

This is sample[9]: 10

Array

```
class ArrayDemo{
    public static void main(String args [ ])
    {
        int [ ] sample = new int [10];
        int i;
        for(i=0;i<10;i++)
            sample[i]=i+1;
        for(i=0;i<10;i++)
        System.out.println(“This is Sample[“ + i + ”]”: + sample[i]);
    }
}
```

Bubblesort

```
class Bubble{
        public static void main(String args[ ] )
        {
          int[ ] nums = {99,-10,100123,18,-978,5623, 463, -9, 287, 49};
          int a,b,t,size;
          size=10;
          System.out.Println(“Original Array is:”);
          for(int i=0;i<size;i++)
              System.out.print(“ “ + nums[i]);
          System.out.println();
         System.out.println(“Sorted array is:”);
```

```
 for(a=1;a<size;a++)
            for(b=size-1;b>=a;b--)
            {
                  if(nums[b-1] > nums[b])
                  {
                        t = nums[b-1];
                        nums[b-1]=nums[b]
                        nums[b]=t;
                  }

            }
System.out.println(" Sorted array is");
for(int i=0;i<size;i++)
      System.out.println(" "+ nums[i]);
System.out.println();
}
```

Multidimenional Array

- **Two dimensional array.**
- **It can be thought of creating a table of data with the data organised by row and column**

int [ ][ ] table= new int [10][20];

O/P:

1 2 3 4

5 6 7 8

9 10 11 12

```
class TwoD{
        public static void main(String args [ ])
        {
                int t,i,k=1;
                int [ ][ ] table = new int [3][4];
                for(i=0;i<3;i++)
                for(j=0;j<4;j++){
                        table[i][j]=k;
                        System.out.println(table[i][j] + " ");
                        k=k+1;
                }
        }
}
```

Irregular Arrays

- When you allocate memory for a multidimensional array, you need to specify only the memory for the first leftmost dimension. Then allocate remaining dimensions seperately

int [ ][ ] table= new int [3][ ];

table[0]=new int [4];

table[1]= new int[4];

table[2]=new int[4];

here no need to allocate the same number of elements for each index.

table[1]=new int [2]

- Arrays of three or more dimensions

  type[][]...[] name = new int [size1][size2[size3].....[sizen];

Assigning array references

- When you assign one array reference variable to another, you are simply changing what object that variable refers to.

```
int i;
  int [] num1 = new int [10];
  int [] num2 = new int [10];
  for(i=0;i<10;i++)
     num1[i]=i;
  for(i=0;i<10;i++)
     num2[i]=-i;
```

Assigning array references

```
System.out.println("Here is num1");
 for(i=0;i<10;i++)
    System.out.println(num1[i]+" ");
 System.out.println();

 System.out.println("Here is num2");
 for(i=0;i<10;i++)
    System.out.println(num2[i]+" ");
 System.out.println();
```

```
 num2=num1;

   System.out.println("Here is num2");
  for(i=0;i<10;i++)
     System.out.println(num2[i]+" ");
   System.out.println();

num2[3]=99;
```

```
System.out.println("Here is num1 after change through num2");
  for(i=0;i<10;i++)
     System.out.println(num1[i]+" ");
   System.out.println();
```

Using length member

```
int [] list = new int [10];
int [] [] table = { {1,2,3},{4,5},{6,7,8,9}};
System.out.println(“Length of list” + list.length);  //
System.out.println(“Length of table” + table.length);  //
System.out.println(“Length of table[1]” + table[1].length); //
for(int i=0;i<list.length;i++)
    list[i]=i*i;
for(int i=0;i<list.length;i++)
    System.out.println(list[i] + “ “);  //
```

for-each style for loop

- It loops through collection of objects such as an array, in strictly sequential fashion from start to finish.

```
for (type itr_var : collection)
    statement - block
```

for-each style for loop

value is: 1

value is: 2

value is: 3

.

.

value is: 9

value is: 10

Summation: 55

for-each style for loop

```
class foreach{
    public static void main(String []args){
        int [] nums = {1,2,3,4,5,6,7,8,9,10};
        int sum = 0;
        for(int x : nums) {
            System.out.println(“value is:”+ x);
            sum=sum+x;
            }
        System.out.println(“Summation :” + sum);
    }
}
```

Strings

- Strings defines and supports character strings.
- In java strings are objects.

```
String str = new String(“Hello”);
String str2 = new String(str);
```

Strings

boolean equals(str) : returns true if invoking string contains the same charcter sequence as str.

int length(): returns no of characters in the string

char charAt(index): returns the character at the index specified by index

int compareTo(str): returns less than zero if the invoking string is less than str, greater than zero if the invoking string is greater than str and zero if strings are equal.

int indexOf(str): searches the invoking string for the substring specified by str. returns the index of the first match or -1 on failure.

int lastIndexOf(str): searches the invoking string for the substring specified by str. returns the index of the last match or -1 on failure.

Example in Netbeans.....

Strings

```
for(int i=0; i<str1.length() ; i++)
        System.out.print(str1.charAt(i));
    System.out.println();
    if(str1.equals(str2))
        System.out.println("Str1 equals str2 :");
    else
        System.out.println("Str1 equals str2");
    result = str1.compareTo(str3);
    if(result == 0)
        System.out.println("Str1 equals str2");
    else if(result<0)
        System.out.println("Str1 < str3");
    else
        System.out.println("Str1 > str3");
```

```
str2 = "one two three one";
        idx = str2.indexOf("one");
        System.out.println("index of first occurance of one:" + idx);
        idx = str2.lastIndexOf("one");
        System.out.println("index of last occurance of one:" + idx);
```

Array of Strings

- Strings can be assembeled into arrays

```
String [] strs = {"this", "is", "a","test"};
        System.out.println("Original Array");
        for(String S:strs)
                System.out.print(S + " ");
```

O/P:

Original Array

this is a test

Modified Array

this was a testtool

Array of Strings

- Strings can be assembeled into arrays

```
String [] strs = {"this", "is", "a","test"};
        System.out.println("Original Array");
        for(String S:strs)
                System.out.print(S + " ");
        System.out.println("\n");
        strs[1]="was";
        strs[3]="testtool";
        System.out.println("Modified Array");
        for(String S : strs)
           System.out.print(S + " ");
```

Strings are immutable

- contents of string objects are immutable.
- string buffer allows to modify string it holds.

```
String orgstr = "Java makes the web move";
        String substr = orgstr.substring(5,18);
        System.out.println("\norgstr:" + orgstr);
        System.out.println("\nsubstr:" + substr);
```

Strings are immutable

- contents of string objects are immutable.
- string buffer allows to modify string it holds.

```
String orgstr = "Java makes the web move";
        String substr = orgstr.substring(5,18);
        System.out.println("\norgstr:" + orgstr);
        System.out.println("\nsubstr:" + substr);
```

O/P:

orgstr:Java makes the web move

substr:makes the web

Using command line arguments

- Command line arguments is the information that directly follow the program name.

```
class CLDemo{
public static void main(String  [] args){
System.out.println(“These are “ + args.length + “Command line arguments”);
System.out.println(“They are”);
for(int i = 0; i<args.length; i++)
        System.out.println(“arg[“+i+”]:” + args [i]);
}
}
```

Introducing classes, objects and methods

- A class is a template that defines the form of an object.
- Objects are instances of a class

Defining a class

```
class classname {
        type var1;
        type var2;
        . .. type varn;
        type method1(params) {
                // body of method
        }
        ....
        type methodn(params) {
                // body of method
        }
}
```

Introducing classes, objects and methods

- Defining a class

```
class vehicle {
        int passengers, fuelcap,mpg;
        }
class vehicleDemo{
        public static void main( String [] args){
        vehicle minivan = new vehicle();
        int range;
        minivan.passengers = 7;
        minivan.fuelcap = 16;
        minivan.mpg = 21;
        range = minivan.fuelcap * minivan.mpg;
System.out.println(“Minivan can carry” + minivan.passengers + “ with a range of “ + range);
                }
}
```

Introducing classes, objects and methods

- Defining a class
- object creation

vehicle car1 = new vehicle();

vehicle car2 = car1

- methods

```
class vehicle {
        int passengers, fuelcap,mpg;
        void range(){
                System.out.println(“range is :” + fuelcap *mpg);
        }
```

```
class vehicleDemo{

        public static void main( String [] args){
         vehicle minivan = new vehicle();

        vehicle sportscar = new vehicle();

              int range1,range2;

        //int range;

        minivan.passengers = 7;

        minivan.fuelcap = 16;

        minivan.mpg = 21;

        sportscar.passengers = 7;

        sportscar.fuelcap = 16;

        sportscar.mpg = 21;

//      range = minivan.fuelcap * minivan.mpg;

//      System.out.println("Minivan can carry" + minivan.passengers + "with a range of" + range);   //“Minivan can carry” + minivan.passengers + //“ with a range of “ + range)

      minivan.range();

      sportscar.range();                }

}
```

Returning from a method

```
int myMath()
{
    for int (i=0;i<10;i++){
        if(i=5) return;
        System.out.println(i);
}
```

Returning a value

```
class vehicle{
        int range(){
                return mpg * fuelcap;
        }
}
class setMath{
        public static void main(String [] args){
        range1=minivan.range();
        range2 = sportscar.range();
        //print the values
}
}
```

using parameters

```
public class Parademo {


    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        checknum e = new checknum();
            if(e.isEven(10))
                    System.out.println("10 is Even");


    }
}
```

using parameters

```
class checknum{
        boolean isEven(int x){
            if((x%2) == 0)
                return true;
            else
                return false;
        }
 }
public class Parademo {
    public static void main(String[] args) {
        // TODO code application logic here
        checknum e = new checknum();
            if(e.isEven(10))
                    System.out.println("10 is Even")
    }
    }
```

Constructors

- **Constructor initializes an object when it is created.**

```
class myclass{
        int x;
        myclass( ) { x=10; }
 }
public class consDemo {
    public static void main(String[] args) {
        // TODO code application logic here
        myclass t1 = new myclass();
        myclass t2 = new myclass();
         System.out.println(t1.x + "  " + t2.x);
    }
    }
```

Parmaterised Constructors

- **Constructor initializes an object when it is created.**

```
class myclass{
        int x;
        myclass(int i) {  x = i;  }
 }
public class consDemo {
    public static void main(String[] args) {
        // TODO code application logic here
        myclass t1 = new myclass(10);
        myclass t2 = new myclass(8);
        System.out.println(t1.x + "  " + t2.x);
    }
    }
```

Adding Constructor to Vehicle class

- Add the code

Adding Constructor to Vehicle class

```
class vehicle {
        int passengers, fuelcap,mpg;
        vehicle(int p, int f, int m)
        {
                passengers=p;   fuelcap=f;  mpg = m;
        }
        }
class vehicleDemo{
        public static void main( String [] args) {
        vehicle minivan = new vehicle(7,6,21);
        vehicle sportscr = new vehicle(2,14,12);
        int range;
                range = minivan.fuelcap * minivan.mpg;
System.out.println("Minivan can carry" + minivan.passengers + " with a range of " + range);
                }
}
```

Adding Constructor to Vehicle class

- Objects are dynamically allocated from a pool of free memory by using new operator.
- Garabage collector reclaims objects automatically.
- It is possible to define a method called a finalizer.

```
protected void finalize( )
{
  // finalization code here
}
```

this keyword

- When a method is called it is automatically passed an implicit argument that is a reference to the invoking object. This reference is called this.