12.19

Arrays

→ Group of like-typed variables that are referenced by common name.
→ May have 1 (or) n dimensions.
→ Elements accessed by index

Advantages

→ Code optimization
  retrieve/sort data easily
→ Random access

Disadvantages

→ Size limit

Array creation (2 parts)

→ Declare a variable of desired array type (1st part)
→ Allocate memory that will hold array using new and assign it to array variable (2nd part)

→ `<datatype> arrayname[]`
  `arrayname = new dt[size]`
  (or) `<dt> a_name[] = new dt[size]`

(eg:) `int monthdays[]` (or) `int[] monthdays` (or)
`int []monthdays` // 3 methods of Array notations

```
class argg
{
    public static void main(String a[])
    {
        double nums[] = {10.1, 10.2, 12.3, 13.4, 14.5}; // (or) {{{ (or)
        double res = 0;
        int i;
        for(i=0; i<5; i++)
            res = res + nums[i];
        System.out.println("Avg. is " + res/5);
    }
}
```

```
double nums[] = new double[5];
nums[0] = 10.1;
nums[1] = 10.2;
nums[2] = 12.3;
nums[3] = 13.4;
nums[4] = 14.5;
```

// Somewhat similar to object declaration

```
for(int i : nums);
```

// No calculation to find length of array is needed.

The entire array is accessed along with increment (no decrement)

No condition.

This for loop is used with arrays and collections.

- Collections → Collection of several datatype variables
  No fixed size. You can dynamically grow the size of it's.

- You can pass arrays as arguments to methods also.

Eg:
```
void mn(int a[])
{
  :
}

int a[] = {1,2,3,4,5};
mn(a);
```

- You can also return arrays in methods.

Built-in methods in Arrays

```
public static void arraycopy(object src, int srcpos, object dest, int dstpos,
```

- arraycopy: In-built method to copy one array to another array
- src: From which array you are going to copy
- srcpos: From which pos. you are going to copy
- dest: To which array you are going to copy
- dstpos: From which pos. you are going to paste

Multi-Dimensional Arrays

```
dt[][] arrayvar;   (or)
dt [][]arrayvar;   (or)
dt arrayvar[][];   (or)
dt []arrayvar[];
```

(2) In 2-d array, it is mandatory to at least mention the row size. (Column size → optional)

(Eg.) `int twod[][] = new int[4][5];`

```
class twodarray
{
  public static void main(String a[])
  {
    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.print();
      }
  }
}
```

`int length);` → Method signature (Prescribed look)

length of the elements in the array you are going to copy

In main(), you are going to use it as `System.arraycopy(....);`

Some more examples:

binarysearch, sort, fill, etc....

fill → to fill elements in array

$$m1 = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 4 & 3 \\ 3 & 4 & 5 \end{bmatrix} \qquad m2 = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 4 & 3 \\ 1 & 2 & 4 \end{bmatrix}$$

```
class matadd
{
  public static void main (String a[])
  {
    int a[][] = { {1,3,4}, {2,4,3}, {3,4,5} };
    int b[][] = { {1,3,4}, {2,4,3}, {1,2,4} };
    int c[][] = new int[3][3];
    for(int i=0; i<3; i++)
      for(int j=0; j<3; j++)
      { c[i][j] = a[i][j] + b[i][j];
        System.out.println(c[i][j]+" ");
      }
      System.out.println();
  }
}
```

Class

Basic for Object-Oriented Programming

Class defines new data type
→ used to create object of that type

Object

instance of class

Syntax

```
class cn
{
dt instance var1 ;
   :
   :
dt instancevarn ;
dt methodname (parameter list)
{ . . . }
}
```

Instance variables (Variables that are going to be accessed by objects)

Variables / data declared within class

Methods and variables defined within class
→ members of class

Declaring objects (Gives memory allocation to class)

1. Declare a variable of class type
2. Must acquire an actual physical copy of obj. and assign it to that variable

Syntax

```
student s = new student()
```

→ constructor (Initialise values to the class variables. Even if user has not initialised any values, compiler assigns default values)

(or)

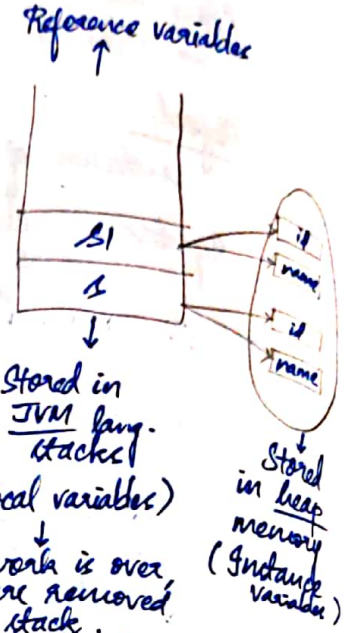
```
student s ;
s = new student() ;
```

new operator

Dynamically allocates memory for an object
→ Allocates memory for an object during run time

```
class student
{
  int id;
  String name;
}
class teststu
{
  public static void main(String a[])
  {
    student s = new student();
    student s1 = new student();
    System.out.println(s.id);
    System.out.println(s.name);
    System.out.println(s1.id);
    System.out.println(s1.name);
  }
}
```

// If a Java code has multiple classes, save the file with the name of the class having main().



Stored in JVM lang. stacks (local variables)

Once work is over, these are removed from stack. (Deallocation)

Stored in heap memory (Instance variables)

Creating multiple objects by one type

```
int i=10, j=20;
student s1 = new student(), s2 = new student();
```

dt — var1 — var2

Anonymous objects (Nameless objects)

Used only at the time of accessing a method (or) instance

Doesn't have a name

Does not create any reference to instance variables in class

Usually:

```
student s = new student();
s.print();
```

Anonymous:

```
new student().print();
```

Methods (Functionality of method → To perform a job)

Syntax

```
type methodname (parameter list)
{
    //Body of method
}
```

2 types : Doesn't return value → return type : void
Returns value → return type : dt

Doesn't return value

```
class recarea
{
    double l, b;
    void rarea()
    {
        System.out.print("Area is ");
        System.out.print(l*b);
    }
}
class testrec
{
    public static void main(String a[])
    {
        recarea r1 = new recarea();
        recarea r2 = new recarea();
        r1.l = 10;
        r1.b = 10;
        r2.l = 20;
        r2.b = 10;
        r1.rarea();
        r2.rarea();
    }
}
```

Returns value

```
// Returning the value
double rarea()
{
    return l*b;
}
```

→ For any dt other than void, the method must contain a return statement.

```
// Calling the method
double a = r1.rarea();
System.out.print(a);
double b = r2.rarea();
System.out.print(b);
```

static Methods (No connection to objects; directly linked to class)

Performs task that doesn't depend on an object
Also called as class methods

Syntax

```
static void test()
{....}
classname.methodname(arg)
```

(Eg) Math class (All methods in this class are public and static)
→ In-built class

abs(x)
ceil(x)
cos(x), sin(x), tan(x)
sqrt(x), pow(x,y), min(), max()

All methods in this class return double dt.

Accessed as
Math.methodname();
Math.PI;

Constants (Associated with Math class)

PI, E // Already defined inside Math class

Random class

Syntax

```
Random r = new Random();
int a = r.nextInt();
```

-2,147,483,648 to +2,147,483,647 // Range of random nos.

Constructors

→ Special type of method used to initialize obj.
→ Automatically initializes obj.
→ It is called constructor because it constructs values at the time of object creation

Rules

→ same name as class in which it resides and syntactically similar to method

→ no explicit return type

→ cannot be abstract, static, final and synchronized // All are keywords

Types

→ Default (no arguments) constructor

→ Parameterized constructor

Implicit return type of class
Constructor is class type itself

// Diff. b/w constructor and other methods

Default constructor

```
classname()
{...}
class init
{
    init()
    {
        System.out.print("Welcome");
    }
    public static void main(String a[])
    {
        init in = new init();
    }
}
```

// Even if the user has not defined a default constructor, compiler automatically invokes a default constructor implicitly where default values are initialised to variables (instance variables)

Parameterized Constructor

Usage : To provide values to distinct objects

```
class student
{
    int id;
    String name;
    student (int i, String n)
    {
        id = i;
        name = n;
    }
    void display ()
    {
        System.out.print(id+" "+name);
    }
    public static void main (String a[])
    {
        student s1 = new student(1, "a");
        student s2 = new student(2, "b");
        s1.display();
        s2.display();
    }
}
```

Roles of constructor apart from initialising values

→ Helps in object creation (object instantiation)
→ When working with threads, constructors are used to start a thread

Constructor overloading

Constructors differing in no. of parameters and types of parameters.

```
class student
{
  int id;
  String name;
  int age;
  student(int i, String n)
  {
    id = i;
    name = n;
  }
  student(int i, String n, int a)
  {
    id = i;
    name = n;
    age = a;
  }
  void display()
  {
    System.out.print(id + " " + name + " " + age);
  }
  public static void main(String a[])
  {
    student s1 = new student(1, "a");
    student s2 = new student(2, "b", 26);
    s1.display();                    // 1 a 0 (default value)
    s2.display();                    // 2 b 26
  }
}
```

To be discussed: Compatibility issues with diff. dt s.

Diff. btw constructors and methods:

Methods	Constructor
Has explicit return type	No explicit return type
Has to be called explicitly	No explicit call. Called whenever an object is declared.
May (or) may not have same class name	Only has class name.
User has to provide method definition explicitly.	Compiler provides default constructor.
Used to perform tasks (or) to expose functionality (behaviour) of objects.	Used for initializing values.

1.2020

Copy constructor

Copies values of one object to another object.

In JAVA, there is no copy constructor. But its functionality is brought about in 3 ways :

→ (Assigning) By constructor

→ (Reference variable) By assigning object values to another

→ Clone method (Purpose and working)

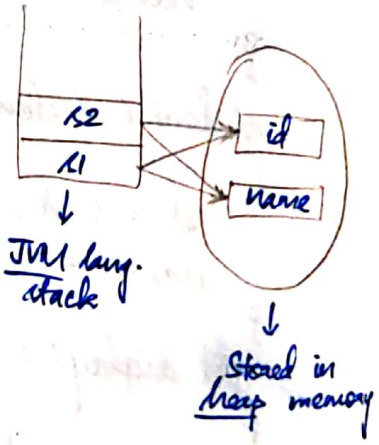
By constructor

```
class student
{
   int id;
   String name;
   student (int i, String n)
   {
      id = i;
      name = n;
   }
   student (student s)
   {
      id = s.id;
      name = s.name;
   }
   void display()
   {
      System.out.println(id + " " + name);
   }
   Public static void main (String args[])
   {
      student s1 = new student (1, "a");
      student s2 = new student (s1);
      s1.display();
      s2.display();
   }
}
```

By assigning object value to another

```
class student
{
  int id;
  String name;
  student (int i, String n)
  {
    id = i;
    name = n;
  }
  student()
  {

  }
  void display()
  {
    System.out.println(id+" "+name);
  }
  Public static void main(String args[])
  {
    student s1 = new student(1, "a");
    student s2 = new student();
    s2.id = s1.id;
    s2.name = s1.name;
    s1.display();
    s2.display();
  }
}
```



Overloading methods in JAVA

→ Define two/more methods within same class that share same name as long as their parameters declared are different.

→ How compiler recognises it

Uses type / no. of arguments as its help to determine which version of overloaded method to actually call

→ Rules (If you don't follow this, compiler faces ambiguity error)

Methods may have different return type (Optional)
Must differ in type / no. of their parameters (Compulsory)

void add (int a, int b)
int add (int a, int b)
↓
leads to ambiguity error

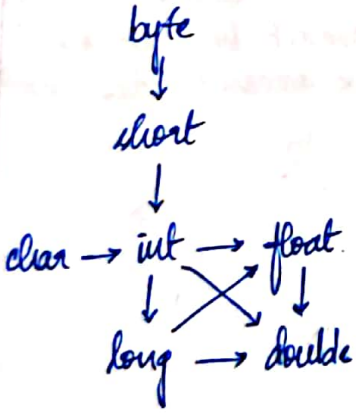
```
class adder
{
  static int add (int a, int b)
  {
    return a+b;
  }
  static int add (int a, int b, int c) //method belongs to class, doesn't need an object to be accessed (class method)
  {
    return a+b+c;
  }
}
class testoverload
{
  Public static void main (String args[])
  {
    add System.out.println (adder.add (10,11));
    System.out.println (adder.add (10,11,12)); // class_name.method_name
  }
}
```

↓
Since you are accessing another class's method.

```
class adder
{
  static int add(int a, int b)
  { return a+b;
  }
  static double add(double a, double b)
  { return a+b;
  }
}
class testoverload
{
  Public static void main(String args[])
  {
    System.out.println(adder.add(10,11));
    System.out.println(adder.add(10.5, 11.5));
  }
}
```

Type promotion / conversion :

(Reverse direction doesn't work)




```
class adder
{ static double add(int a, int b)
  { return a+b;
  }
  static int add(int a, int b)
  { return a+b;
  }
}//
```

```
// class testoverload
{
   Public static void main (String args[])
   {
      System.out.println (adder.add(10,11)) ; // Ambiguity error
   }
}
```

Can main() method be overloaded ?

It can be overloaded in recent versions of JAVA.

JVM treats main() as the entry point only if it has an array parameter of type String.

```
main (String a[])
main (double d[])
```

Access modifiers

- → private
- → protected
- → public
- → default

private

- → Accessible only inside the class / method.
- → Scope is limited to that class / method.
- → Constructor can't be private.

```
class a
{ private int a2 = 40 ;
}
class b
{
   Psvm (String a[])
   { a a1 = new a() ;
     S.o.p(a1.a2) ;   // This leads to error
   }
}
```

protected

→ In order to protect the data of any one class during inheritance by another class.

public

→ Accessed anywhere

default

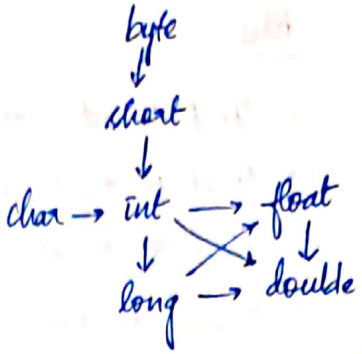
→ default access modifier

Access modifier	Class	Within package	Outside package by sub-class	Outside package
private	Y	N	N	N
default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

7.1.2020

Method overloading :

// If (int a, long b) and (long a, int b) are there and (20, 20) is given, it results in ambiguity error as there is no exact match.




```
class overloadingeg
{                          int a  long b
    void sum (int a, int b)
    {
        System.out.println ("int arg. method");
    }                      long a  long b  long c
    void sum (long a, long b)
    {
        System.out.println ("long arg. method");
    }
    public static void main (String a[])
    {
        overloadingeg ol = new overloadingeg();
        ol.sum (20, 20);   // Here, exact match is available.
                           // ∴ int arg. method is called.
    }
}
```

// In this case, there is no exact match. ∴ int arg. method is called due to type promotion. If (20L, 20) is passed, then it is an error since long can't be depromoted to int.

this keyword

→ When a local variable has same name as instance variable, it hides the instance variable.

→ this keyword directly refers to object

→ Resolves name space collisions that might occur between instance and local variables.

(PTO)

this :

→ Used to refer current instance variable.
→ Used to refer current class method (implicitly).
→ Used to invoke current class instructor.
→ Passed as argument in method call.
Passed as argument in constructor call.
→ You can also return a value.

this is used as a constructor so as to reuse the constructor.

To refer instance variables:

```
class student
{
    int rollno;
    String name;
    float fee;
    student(int rollno, String name, float fee)
    {
        this.rollno = rollno;
        this.name = name;
        this.fee = fee;
    }
    void display()
    {
        System.out.println(rollno+" "+name+" "+fee);
    }
}
class testthis
{
    public static void main(String args[])
    {
        student s1 = new student(111, "a", 5000F);
        student s2 = new student(20, "b", 6000F);
        s1.display();
        s2.display();
    }
}
```

// If you remove this keyword here, the default values are displayed since no values are stored in object.

If same names are not used for instance and local variable this keyword need not be used

To refer class method :

```
class a
{
    void m()
    {
    }
    void n()
    {
      m();          // this.m() is implicitly used by the compiler
    }
    public static void main(String a[])
    {
      a al = new a();
      al.n();
    }
}
```

implicitly adds this keyword automatically

This means the same object used to access n() is used to access m().