One Dimensional Array And Methods

Passing Individual Array Elements to a Method

We know that an array element is treated as any other simple variable in the program. So we can pass individual array elements as arguments to a Method like other simple variables.

Program to check whether individual array element is Even or Odd

```
import java.util.*;
 class  IndividualArrayElementAsArgument
  {
   public static void main (String[] args )
  {
    Scanner s = new Scanner(System.in);
     int[] arr = new int[5];
      int j,i;
     for(i=0;i<arr.length;i++)
       {
          System.out.printf ("Enter the value for arr[ %d] :\n" ,i);
           arr[i] = s.nextInt();
             check(arr[i]);
          }
    } //main

static void check(int m)
  {
     If(m%2 == 0)
       System.out.printf("%d is even",m);
     else
       System.out.printf("%d is  odd",m);
  } //check
} //class
```

Passing whole I-D Array to a Method

We can pass whole array as an actual argument to a method. The corresponding formal argument should be declared as an array variable of the same data type.

```
public static void main(String[] str)
 {
 Int[]  arr = new int[10]
……..
……..
func(arr);          /*In method call, array name is specified without brackets*/
}
```

```
static void func(int[]  ar) {
……..
……..
}
```

We have studied that changes made in formal arguments do not affect the actual arguments, but this is not the case while passing an array to a method. **The mechanism of passing an array, to a method is quite different from that of passing a simple variable.** We have studied earlier that in the case of simple variables, the called method creates a copy of the variable and works on it, so any changes, made in the method do not affect the original variable. **When an array is passed as an actual argument, the called method actually gets access to the original array and works on it, so any changes made inside the method affect the original array.**

Program to understand the effect of passing an array to a method

```
class  SumOfSquareOfArrayElements
 {
  public static void main (String[] args )
 {
     int[] arr = {1,2,3,4,5};
    int i;
   func(arr);
  System.out.println("After making changes the original array elements are :");
    for(i=0;i<arr.length;i++)
      {
        System.out.print (arr[i] + "  " );

        }
  } //main

static void func(int[] val)
 {
 int sum=0,i;
 for(i=0;i<val.length;i++)
  {
   val[i]=val[i]*val[i] ;
  sum+=val[i];
  )
System.out.printf("The sum of squares = %d\n",sum);
} //fun
} //class
```

Here we can see that the changes made to the array inside the called function are reflected in the calling function. The name of the formal argument is different but it refers to the original array.

Write a program using arrays, that reads a decimal number and converts it to (1) Binary (2) Octal or (3) Hexadecimal depending on user's choice.

```
import java.util.*;
 class  Conversion
  {
   public static void main (String[] args )
  {
    Scanner s = new Scanner(System.in);
    int num, opt;
    System.out.println ("Enter a decimal number ") ;
    num = s.nextInt();
   System.out. printf ("1. :Binary\n 2. Octal\n 3. Hexadecimal \n" );
    System.out.println ("Enter your option ") ;
    opt = s.nextInt();
   switch(opt)
    {
          case 1:
                    System.out.println ("Binary equivalent is ");
                    func(num,2);
                    break;
          case 2:
                   System.out.println ("Octal equivalent is ");
                    func (num, 8) ;
                   break;
          case 3:
                   System.out.println ("Hexadecimal equivalent is ") ;
                    func (num, 16);
                   break;
   }//switch
} //main

static void func(int num, int b)
  {
   int i=0, j ,rem;
   char arr[20];
   while(num>0)
    {
     rem=num%b;
    num/=b;
    if(rem>9&&rem<16)
          arr[i++]=rem-10+'A' ;
    else
          arr[i++]=rem+'0' ;
```

```
   }
   for(j=:i-l;j>=0;j- -)
    System.out. printf( "%c", arr[j]);
} // func

}//class
```

Write a program to sort array elements using method.

```
class SortNumbers
{
    public static void main(String[] args)
    {
        int[] data={40,50,10,30,20,5};
        System.out.println("Unsorted List is :");
        display(data);
        sort(data);
        System.out.println("\nSorted List is :");
        display(data);
    } //main
    static void display(int num[])
    {
      for(int i=0; i<num.length;i++)
         System.out.print(num[i] + " ");
    } // dispaly
    static void sort(int num[])
    {
     int i, j, temp;
     for(i=0; i<num.length-i;i++)
        {
           for(j=0; j<num.length-i-1;j++)
              {
                 if(num[j]>num[j+1])
                 {
                    temp = num[j];
                    num[j] = num[j+1];
                    num[j+1] = temp;
                 }
              }
        }
    } // sort
}
```

Returning Array from Methods

Java doesn't support multi-value returns.
If all returned elements are of same type, we can return an array in Java. A reference to an array can be returned from a method

```
public class ReturnArray
{
  public static void main(String[] args)
  {
    final int NUMBER = 4;

    // Values reference the array returned from the randomArray method.
    values = randomArray(NUMBER);

    // Display the values in the array.
    for (int i = 0; i < values.length; i++)
    {
      System.out.print(values[i] + " ");
    }
  }


  public static int[] randomArray(int n)
  {
    int[] list = new int[n];

    for (int i = 0; i < list.length; i++)
    {
      list[i] = (int) (Math.random() * 10);
    }

    return list;
  }
}
```

Object and Method

Passing object as parameter:

We can pass Object of any class as parameter to a method in java. We can access the instance variables of the object passed inside the called method. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

```
class Rectangle {
   int length;
   int width;

   Rectangle(int l, int b) {
      length = l;
      width = b;
   }

   void area(Rectangle r1) {
      int areaOfRectangle = r1.length * r1.width;
      System.out.println("Area of Rectangle : "
                         + areaOfRectangle);
   }
}

class RectangleDemo {
   public static void main(String args[]) {
      Rectangle r1 = new Rectangle(10, 20);
      r1.area(r1);
   }
}
```

Myth: "Objects are passed by reference, primitives are passed by value"

Truth #1: Everything in Java is passed by value. Objects, however, are never passed at all.

That needs some explanation - after all, if we can't pass objects, how can we do any work? The answer is that we pass references to objects.

That sounds like it's getting dangerously close to the myth, until you look at truth #2:

Truth #2: The values of variables are always primitives or references, never objects.

Although Java is strictly pass by value, the precise effect differs between whether a primitive type or a reference type is passed.

When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference. Basically, a parameter cannot be changed by the function, but the function can ask the parameter to change itself via calling some method within it.

```
class ObjectPassDemo
{
int a, b;
ObjectPassDemo(int i, int j)
{
a = i; b = j;
}
// return true if o is equal to the invoking  object notice an
//object is passed as an  argument to method
boolean equalTo(ObjectPassDemo o)
{
return (o.a == a && o.b == b);
}
}
// Driver class
public class Test
{
public static void main(String args[])
{
ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);
System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));
System.out.println("ob1 == ob3: " + ob1.equalTo(ob3));
}
}
```

Passing array of objects:

```
class Employee
{
 int empno;
 String name;
String city = "Indore";
 Employee() {  }
 Employee(int e, String n)
  {
    empno=e;
    name=n;
  }
  void display()
   {
    System.out.println("Employee no :"+empno);
    System.out.println("Employee name :"+name);
    System.out.println("Employee city :"+city);
   }
}
class PassArrayOfObjects
  {
   static void argumentAsArrayOfObjects(Employee[] e)
         {
          for(Employee e1 : e)
             e1.display();
         }
   public static void main(String[] str)
            {
    Employee[] e = {        new Employee(100,"Neeraj"),new Employee(101,"Ramesh"),
                            new Employee(102,"Suresh"),new Employee(103,"Ajay"),
                            new Employee(104,"Shubham")  };
 argumentAsArrayOfObjects(e);
           } //main
} //class
```

Returning Object from method:

A method can return any type of data, including class types that you create. For example, in the following program, the getRectangleObject() ( ) method returns an object of Rectangle.

```
class Rectangle {
    int length;
    int breadth;

    Rectangle(int l,int b) {
     length = l;
     breadth = b;
    }

    Rectangle getRectangleObject() {
     Rectangle rect = new Rectangle(10,20);
     return rect;
    }
}

class RetOb {
    public static void main(String args[]) {
     Rectangle ob1 = new Rectangle(40,50);
     Rectangle ob2;

     ob2 = ob1.getRectangleObject();
     System.out.println("ob1.length : " + ob1.length);
     System.out.println("ob1.breadth: " + ob1.breadth);

     System.out.println("ob2.length : " + ob2.length);
     System.out.println("ob2.breadth: " + ob2.breadth);

     }
}
```

```
class Employee{
 double salary;
 Employee(double salary){
  this.salary = salary;
 }
 Employee updateSalary(double salary){
  Employee employee = new
Employee(this.salary+salary);
  return employee;
 }
 double getSalary(){
  return this.salary;
 }
}
class ReturnObjectDemo{
 public static void main(String args[]){
  Employee kallis = new Employee(34029.48);
  Employee ronaldo;
  ronaldo=kallis.updateSalary(6295.28);
  System.out.println("Salary of Kallis is:
"+kallis.getSalary());
  System.out.println("Salary of Ronaldo is:
"+ronaldo.getSalary());
 }
}
```

Returning Array of Object from method:

```
class Employee
{
 int empno;
 String name;
String city = "Indore";
 Employee() {  }
 Employee(int e, String n)
  {
    empno=e;
    name=n;
  }
  void display()
   {
    System.out.println("Employee no :"+empno);
    System.out.println("Employee name :"+name);
    System.out.println("Employee city :"+city);
   }
}

class ReturnArrayOfObjects
{
 static Employee[] returnArrayOfObjects()
    {
     Employee[] e = {       new Employee(100,"Neeraj"),new Employee(101,"Ramesh"),
                             new Employee(102,"Suresh"),new Employee(103,"Ajay"),
                            new Employee(104,"Shubham")  };
     return e;
    }
  public static void main(String[] str)
  {
   Employee[] e1 = returnArrayOfObjects();
   for(Employee e2: e1)
    e2.display();
   }
}
```