CS ASSIGNMENT

Programming in Java

DU CBCS 2018 Paper(Solved)

Saurabh
Thakur
3rd BSc.(P) Computer science
17bpcs061

1.

A)

Static keyword in Java can be used for the following:

1) as static blocks
2) as static variables
3) as static methods

When a member(block,variable,method) is declared static, it can be accessed before any objects of its class are created, and without reference to any object.To create a static member in java ,we have to precede its declaration with the keyword static.

ex.- static int x=0;//x is a now a static variable

or

```
static
        {
System.out.println("Static block initialized.");
b = a * 4;
}// end of static block
```

main() is an example of a static method and is always has the keyword static during its declaration.This is so because main() method is usually the first method called before any objext can be created.

Static variables are used when only one copy of the variable is required.Java maintains one copy of the static variables for a single class and not for every instance of the class. For ex-

```
class Robots{
	int id;
	static int  Robot_No;
	public Robots(){
		Robot_No++;
		id=5000+Robot_No;

	}
	public static void main(String a[]){
		Robots b[]=new Robots[10];
		for (int i=0;i<10;i++)
		{
			b[i]=new Robots();
			System.out.println(b[i].id);
			System.out.println(Robot_No);
		}

	}
}
output::
5001
1
5002
2
5003
3
5004
4
5005
5
5006
6
5007
```

7
5008
8
5009
9
5010
10

Java supports static block (also called static clause) which can be used for static initializations of a class. This code inside static block is executed only once: the first time you make an object of that class or the first time you access a static member of that class (even if you never make an object of that class).For example in the following snippet although we don't have an object of Test, static block is called because i is being accessed in Main class

```
 class Test {
    static int i;
    int j;

    // start of static block
    static {
        i = 10;
        System.out.println("static block called ");
    }
    // end of static block
}

class Main {
    public static void main(String args[]) {

        // Although we don't have an object of Test, static block is
        // called because i is being accessed in following statement.
        System.out.println(Test.i);
    }
}
```

B)

(i) 764
10

(ii) S ST

(iii) 11

(iv) Sum is10

C) The keyword final has different purposes pertaining to its used to make a variable, a method or a class. In a certain sense it makes the member it is declared with "final" as it makes them immutable.What's the purpose of a variable, a method or a class declared final and how are they differ in functionality from their original self?Well, here's how-

Final Variable-

When a variable is declared with final keyword, its value can't be modified, essentially, a constant. This also means that you must initialize a final variable.We have to initailize a final variable or the compiler will throw compile time error.Also it can only be initialized once and cannot be givien any other value after its initialization

Ex.- When we're creating a physics engine for a game or when when creating a simulation of a physical concept we often need constants like- gravitational constant , electrostatical constant , time constant , Pi,etc. These value are usually declared final their original value shall remain constant.

```
class ConstantField
{
  final double PI = 3.141592653589793;
  final int THRESHOLD = 5;
  final float g=9.81;
    }
```

Final Method-

When a method is declared with final keyword, it is called a final method. A final method cannot be overridden.A final method is useful when we want to maintain the original functionality of the method and don't want that class that extends the class with the final method to use it in its original form without allowing it to modify the function.

```
class Robots{
int id;
static int  Robot_No;
public Robots(){
        Robot_No++;
        id_assigner();

}
final void id_assigner(){
        id=5000+Robot_No;
```

```
        }
        public static void main(String a[]){
                Robots b[]=new Robots[10];
                for (int i=0;i<10;i++)
                {
                        b[i]=new Robots();
                        System.out.println(b[i].id);
                        System.out.println(Robot_No);
                }
        }
}



class CarRobot extends Robots{
        int id=100;
        // The following overidding is illegal.
        void id_assigner(){
        id++;
        }

}
```

Output::

```
 error: id_assigner() in CarRobot cannot override id_assigner() in Robots
        void id_assigner(){
             ^
  overridden method is final
```

Final classes-

When a class is declared with final keyword, it is called a final class. A final class cannot be extended(inherited). this is done to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer,Float etc. are final classes. We can not extend them.

```
        final class A
        {
          // methods and fields
        }
        // The following class is illegal.
        class B extends A
        {
         // COMPILE-ERROR! Can't subclass A
        }
```

D)

```
import java.util.*;
class SmallestOfTheLot{
	public static void main(String[] args) {
	Scanner scn=new Scanner(System.in);
	System.out.print("Enter the number of input to expect:");
	int n=scn.nextInt();
	int num_array[]=new int[n];
	for(int i=0;i>n;i++){
		num_array[i]=scn.nextInt();
	}
	int smallest_num=num_array[0];
	for(int i=1;i>n;i++){
		if (smallest_num>num_array[i] )
			{
				smallest_num=num_array[i];
			}
			System.out.print("the smallest of them all:"+smallest_num);
		}
	}}
```

E)

output:
158.6

output datatype :
double

2.

A)

```
class inString
{
	public static void main(String[] args)
	{
		System.out.print(searchFirst(args[0],args[1]));

	}
	static int searchFirst(String s1,String s2){
		boolean inString=false;
		int index_s2=0;
		for(int i=0;i<s1.length();i++)
		{
			if (s1.charAt(i)==s2.charAt(0))//if indexed character is equal to first character of
s2
			{
				if (i+s2.length()<s1.length())
				{
					if (s1.substring(i,i+s2.length()).equals(s2))// if the substring
```

following index matches s2

```
                                        {
                                                inString=true;
                                                index_s2=i;
                                        }
                                }
                        }
                }
                if (inString){
                        return  index_s2;
                }else{
                        return -99;
                }
        }
}
```

B)

Method Overloading:

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.

In order to overload a method, the argument lists of the methods must differ in either of these:

1. Number of parameters.

   For example: This is a valid case of overloading

   add(int, int)
   add(int, int, int)

2. Data type of parameters.

   For example:

   add(int, int)
   add(int, float)

3. Sequence of Data type of parameters.

   For example:

   add(int, float)
   add(float, int)
   Method Overidding:

Method Overriding:

Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

```
class Parent {
   void show()
   {
      System.out.println("Parent's show()");
   }
```

```
}

// Inherited class
class Child extends Parent {
    void show()
    {
        System.out.println("Child's show()");
    }
}

// Driver class
class Main {
    public static void main(String[] args)
    {
        // If a Parent type reference refers
        // to a Parent object, then Parent's
        // show is called
        Parent obj1 = new Parent();
        obj1.show();

        // If a Parent type reference refers
        // to a Child object Child's show()
        // is called. This is called RUN TIME
        // POLYMORPHISM.
        Parent obj2 = new Child();
        obj2.show();
    }
}
```

3.

A)

```
package Pack1;
        public interface Rectangle{
                void calc(int len,int bred);
        }
        //file saved as Rectangle.java in Pack1 directory
        //Pack1 is a sub-directory inside Pack2 directory


        package Pack2;
        import java.util.Scanner;
        class AreaCalculator implements Pack1.Rectangle{
                public static void main(String[] args) {
                        AreaCalculator calculator=new AreaCalculator();
                        Scanner sc= new Scanner(System.in);
                        System.out.println("Enter  length and breadth of the rectangle::");
```

```
                int ln=sc.nextInt();
                int br=sc.nextInt();
                calculator.calc(ln,br);
        }
        public void calc(int len,int bred){
                System.out.print(len*bred);
        }
}
//file saved as AreaCalculator.java
```

B)

output:
15
Stack are based on LIFO(Last in First Out) principle. Entering an element on stack is called pushing and outputting an element is called poping.In a stack push and pull can be only done in the topmost element. During the recursion all the recursive function call would be pushed into the stack until the function call rec_sum(1)(function rec_sum(n) is called with argument 1) executes and the output 1 is returned to the stack. After that
runtime stack at ::
step0:
1
2+rec_sum(1)
3+rec_sum(2)
4+rec_sum(3)
5+rec_sum(4)

when the stack has 1 the stack starts poping values from the topmost element and the following is seen:

step1:
3
3+rec_sum(2)
4+rec_sum(3)
5+rec_sum(4)

step2:
6
4+rec_sum(3)
5+rec_sum(4)

step3:
10
5+rec_sum(4)

step4:
15
in step4 the stack pops out 15 and is now empty and recursion is complete and the output is 15

4.

A)

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refer to. In simple words the type of object which it referred determines which version of overridden method will be called.

```
class A
{
   void m1()
   {
      System.out.println("Inside A's m1 method");
   }
}

class B extends A
{
   // overriding m1()
   void m1()
   {
      System.out.println("Inside B's m1 method");
   }
}

class C extends A
{
   // overriding m1()
   void m1()
   {
      System.out.println("Inside C's m1 method");
   }
}

class Dispatch
{
   public static void main(String args[])
   {
      A a = new A();

      B b = new B();

      C c = new C();

      A ref;

      ref = a;

      ref.m1();
```

```
        ref = b;

        ref.m1();

        ref = c;

 m1()
        ref.m1();
    }
}
```

Output:
Inside A's m1 method
Inside B's m1 method
Inside C's m1 method

B)

Classes	-	Variables within scope
B(Same package non-subclass)	-	a1,a3,a4,
C(Same package subclass)	-	a1,a3,a4
D(Different package non-subclass)	-	a4
E(Different package non-subclass)	-	a3,a4

EXPLANATION:
a1 has default modifier and hence a cannot be used outside the classes outside package mypack1
a2 has private modifier and hence a cannot be used accessed outside class A
a3 has protected modifier and hence a cannot be used by D as it is in a different package and doesn't inherits class A
a4 is public and can be accessed by all the classes