

Q1. Short answers to the given questions are as follows:

a. Why Java is called platform independent?

Java is called platform independent as we can run Java codes in any machine that is, the java compiled codes (byte code) can run on all operating systems. The java source code is compiled by the java compiler first, which compiles this code into a bytecode. The Java run-time system, which is called the Java Virtual Machine (JVM) which is an interpreter, recognizes the platform it is on and interprets the bytecode into a machine code which is why java is called a platform independent language.

b. What does a constructor do? What are the syntactic differences between a constructor and a method?

In Object Oriented Programming, constructors are used to initialize objects and constructors do not return any value.

Syntactic differences between a constructor and a method:

- Constructors initialize objects that are created with the “new” operator and on the other hand, Methods perform operations on already existing objects.
- Constructors are required to be the same name as the class name but constructors can't return anything and on the contrary, Methods must be declared to return something, although it can be void.

c. What is access modifier? Mention each of the modifiers scope.

Access modifiers specify which classes can access a given class and its fields, constructors and methods. Access modifiers are used to set the visibility of classes, interfaces, variables, methods, constructors, data members, and the setter methods.

In Java, there are four types of Java access modifiers:

1. Private:

Scope is only within the class. It cannot be accessed from outside the class.

2. Default:

Scope is only within the package. It cannot be accessed from outside the package. If we do not specify any access level, it will be the default one.

3. Protected:

Scope is within the package and outside the package through child class. If child class is not made, it cannot be accessed from outside the package.

4. Public:

Scope is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

d. What are the differences between a class, abstract class and an interface?

In OOP, a Class is like a blueprint for creating objects.

Abstract classes are used to provide some common functionality across a set of related classes while also allowing default method implementations. An abstract class cannot be instantiated, which means it is not allowed to create an object of it.

Interface in Java is a mechanism to achieve abstraction and multiple inheritance. There can be only abstract methods in the Java interface but no method body.

Difference among these three:

Classes can implement multiple interfaces, but only one abstract class. Abstract classes can contain non-abstract methods. They can both have methods, variables, and neither one can be instantiated. All variables declared in an interface are final, while an abstract class may contain non-final variables.

e. Give example of method overriding and method overloading?

Method overloading is the example of compile time polymorphism while method overriding is the example of run time polymorphism.

Java Method Overloading example:

```
class OverloadingExample
{
    static int add(int a,int b)
    {
        return a+b;
    }
    static int add(int a,int b,int c)
    {
        return a+b+c;
    }
}
```

Java Method Overriding example

```
class Parents
{
    void fun()
    {
        System.out.println("Having Fun!");
    }
}
```

```
}  
class Son extends Parents  
{  
void fun()  
{  
System.out.println("Having more fun playing!");  
}  
}
```

f. Why should you use abstract class in your program?

Abstract classes are used to provide some common functionality across a set of related classes while also allowing default method implementations. An abstract class cannot be instantiated (which means it is not allowed to create an object of it) and is meant to be subclassed.

g. Compiler and interpreter- why both is needed in java?

Name of Java Compiler is Javac (Included in the JDK)

Name of Java Interpreter: Java

Both compilers and interpreters are used in java to make it platform independent. That is, the source code is first compiled to a binary byte code and then JVM (Java Virtual Machine, an interpreter) interprets the byte code and thus making the java code platform independent.

h. Why multiple inheritance is not allowed in Java? How java solves this problem?

Multiple inheritance is not allowed in Java to prevent ambiguity. Let us consider a case where class B extends class A and Class C and both class A and C have the same method display(). In such cases, java compiler cannot decide which display method to inherit. To prevent such situation from happening multiple inheritances is not allowed in java.

So java solves this problem using interface. In java, one class can implement two or more interfaces. This also does not cause any ambiguity because all methods declared in interfaces are implemented in class.

i. What is an exception? Why do we need to handle exception? List out some Runtime Exceptions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object is called an exception object which contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called throwing an exception.

The method of handling exceptions is try-catch. We are required to put the code we want to run in the try block, and any exceptions that the code throws are caught by one or more catch blocks. This method will catch any type of exceptions that get thrown. This is the mechanism for handling exceptions.

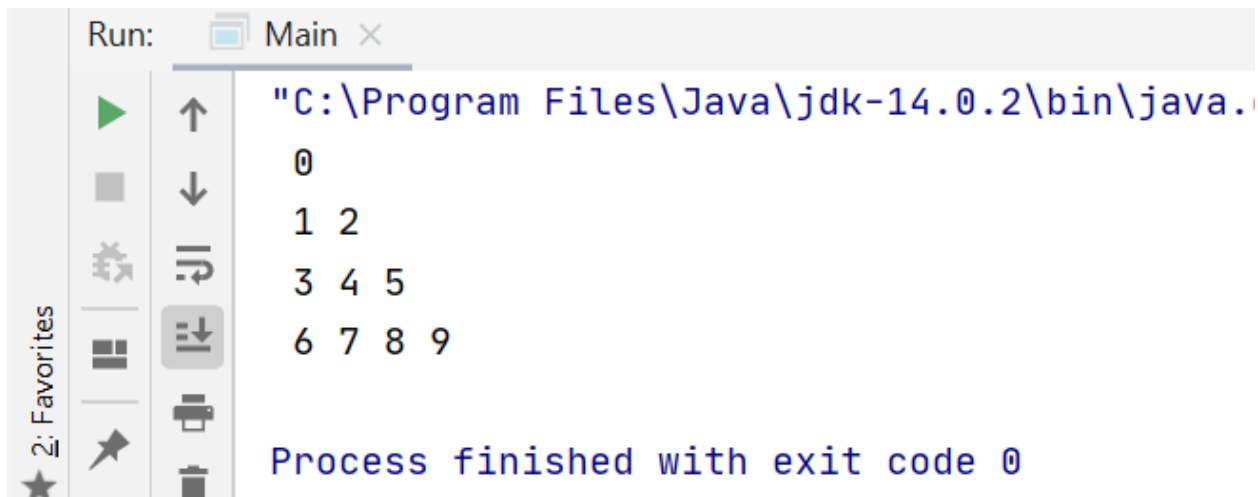
A list of some runtime exceptions is given below:

- ArithmeticException

- NullPointerException
- NumberFormatException
- IndexOutOfBoundsException

Q2. i. Compiling the given java programs and generating the Output of the given code segments and stating the reasons:

Output:



```
Run: Main x
"C:\Program Files\Java\jdk-14.0.2\bin\java.
0
1 2
3 4 5
6 7 8 9
Process finished with exit code 0
```

Fig 01: Snapshot of the output generated by the given code.

Reason:

According to the code given,

It is seen that a two dimensional integer array is being declared which has been specified to have 4 rows. The next 4 lines then defines the column numbers in each of the rows respectively. Two loops that is, nested loops are used to traverse through the array where there is an outer loop with the variable 'i' as an iterator

and an inner loop with a variable 'j' used as an iterator. The outer loop is meant to indicate the row while the inner loop is meant to indicate the column. 'i' iterates from 0 to 3 (i. e the four rows of the given array 'arr'). For each row, the inner loop iterates from 0 to 'i + 1' as each row has 'row + 1' number of columns. These two loops iterate and assign value 'k' to a cell and then increases 'k' to 'k + 1'.

```
int i, j, k = 0;
for (i = 0; i < 4; i++)
{
    for (j = 0; j < i + 1; j++)
    {
        arr[i][j] = k;
        k++;
    }
}
```

The above code snippet indicates that for every value of i, j iterates up to i times. The value of k is initially set to k=0. Every time the inner loop runs, the value of k increases by 1 and in this way, inside the inner loop, value of each of the cells of the given array is being assigned. The next code snippet then demonstrates how the array contents are printed.

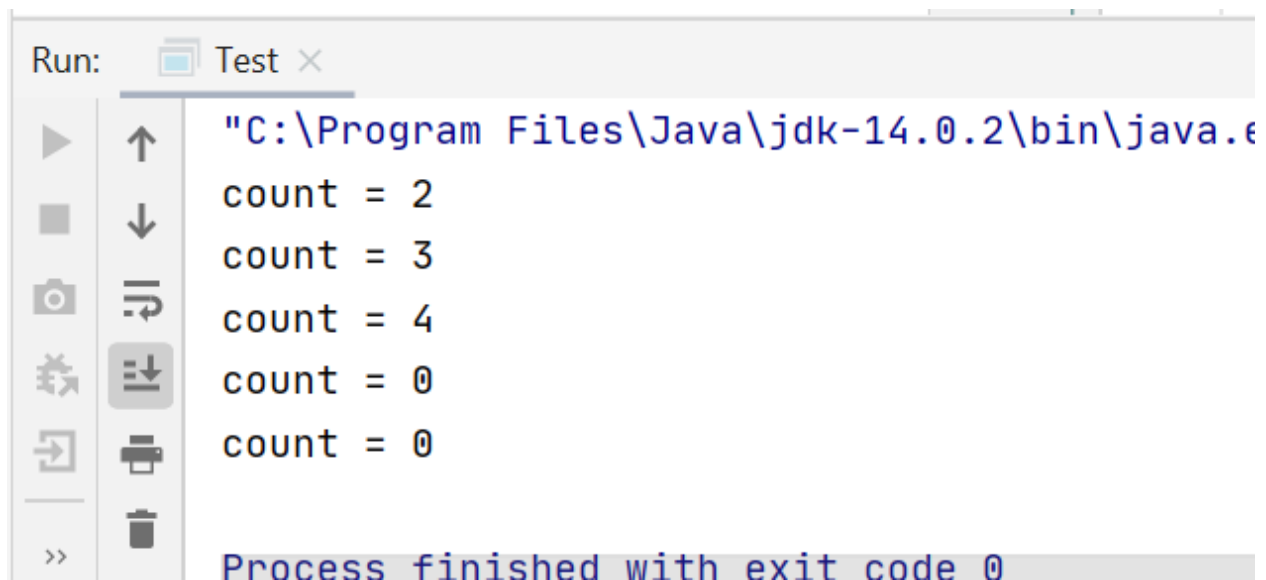
```
for (i = 0; i < 4; i++) {
    for (j = 0; j < i + 1; j++) {
        System.out.print(" " + arr[i][j]);
        k++;
    }
    System.out.println();
}
```

In the above code snippet, nested loops are again used to print the contents of the cell in a pattern. Two loops are used to print the values of the array. The outer loop iterates through each of the rows and the inner loop iterates through each of the

columns and prints the values of the cells. A newline is printed after every row that is, after the inner loop is executed. This is the reasoning behind how the output is being generated.

Q2. ii. Compiling the given java programs and generating the Output of the given code segments and stating the reasons:

Output:



```
Run: Test x
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe"
count = 2
count = 3
count = 4
count = 0
count = 0
Process finished with exit code 0
```

Fig 02: Snapshot of the output generated by the given code.

Reason:

In the given code, the integer variable 'count' is static in nature which means, this variable is shared by all objects of the class and can be accessed directly by using the name of the class. The static variable is being initialized with 0. There is a constructor present which increases the value of static 'count' by 1 each time it is called. Therefore, every time we are creating new objects of 'Test' class, we are calling the constructor 'Test()' which is increasing the value of count by 1.

When object t1 was created, the value of count became 1. Then the value of count was increased and became 2 when another object t2 was created. The value of t1.count is then printed which is 2. This happened because all objects share the same static variable. Every time an object is created, count increases and it becomes the count of every object. So, t1.count is 2 and not 1. In the next line, value of t1.count was increased by 1 so its value became 3. As the static variable count is shared by all objects, the value of count happens to be, count = 3, when the value of t2.count is printed. Again the count is increased using t2.count++ and it becomes 4. So, count = 4 is printed directly using the class name.

Finally, count is being reset to zero and printed again as t1.count and t2.count. Thus, we get count = 0 as the output of both the fourth and fifth lines of the printed code which is the reasoning behind the given code snippet.

Q2. iii.(a) To Generate the output of the given code

Output:

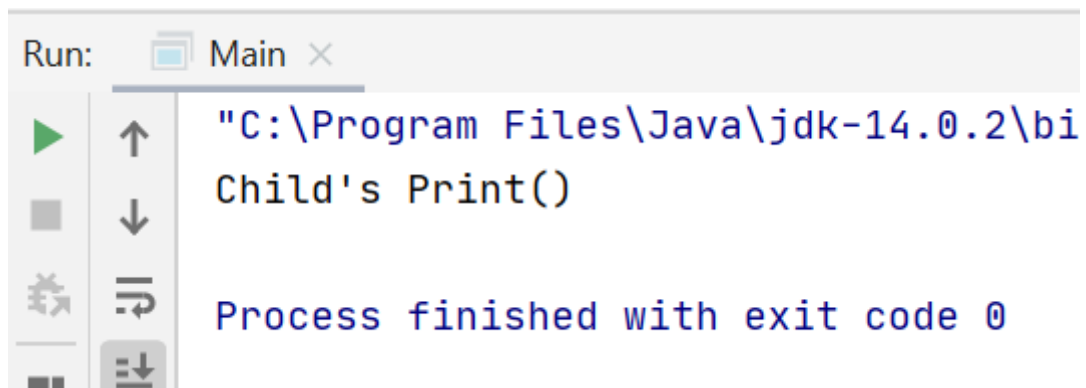


Fig 03: Snapshot of the output generated by the given code.

Q2. iii.(b) To Modify the program so that the output becomes:

Grandparent's Print()

Parent's Print()

Child's Print()

Modified Source Code:

```
class Grandparent
{
    Grandparent() {
        System.out.println("Grandparent's Print()"); }
}

class Parent extends Grandparent
{
    Parent() {
        System.out.println("Parent's Print()"); }
}

class Child extends Parent{
    Child() {
        System.out.println("Child's Print()"); }
}

public class Main
{
    public static void main(String[] args) {
        Child c = new Child(); }
}
```

Output after code modification:

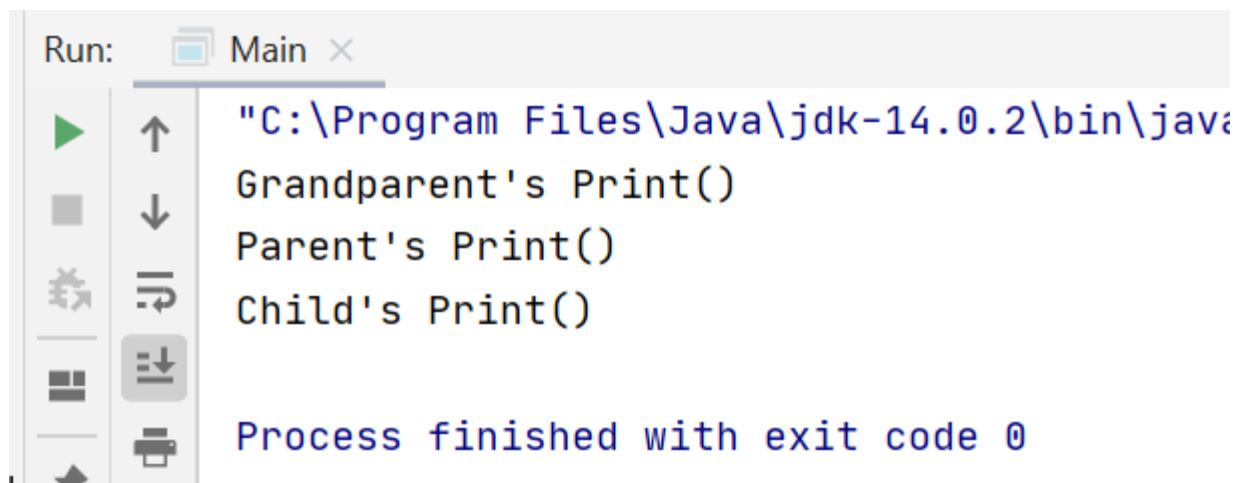
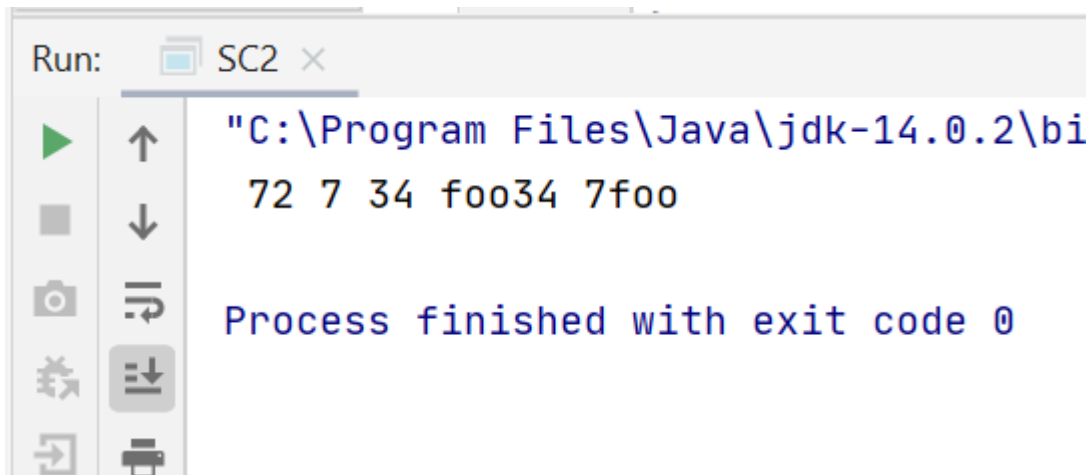


Fig 04: Snapshot of the output generated by the modified code.

Q2. iv. Compiling the given java programs and generating the Output of the given code segments and stating the reasons:

Output:



```
Run: SC2 x
"C:\Program Files\Java\jdk-14.0.2\bin
72 7 34 foo34 7foo
Process finished with exit code 0
```

Fig 05: Snapshot of the output generated by the given code.

Reason:

According to the given code,

First print statement:

Prints a space, '7', '2' and another space after it. Here, the numbers 7 and 2 are considered characters because a space is printed first. So, the output looks like "72".

Second print statement:

Prints the summation of a and b, which were previously initialized as a=3 and b=4. Therefore, the output looks like "7"(i. e the summation).

Third print statement:

Prints a space, the value of a, the value of b followed another space. Here, the values of a and b are not added before printing because a space is printed first which is why the values of a and b are separately considered tokens of the output string. The output is "34".

4th print statement:

This print statement calls foo() method, which returns the string "foo", and prints the value of a, b followed by a space. Here, the values of a and b are considered to be separate tokens of the output string as the string "foo" was printed first. The output is "foo34".

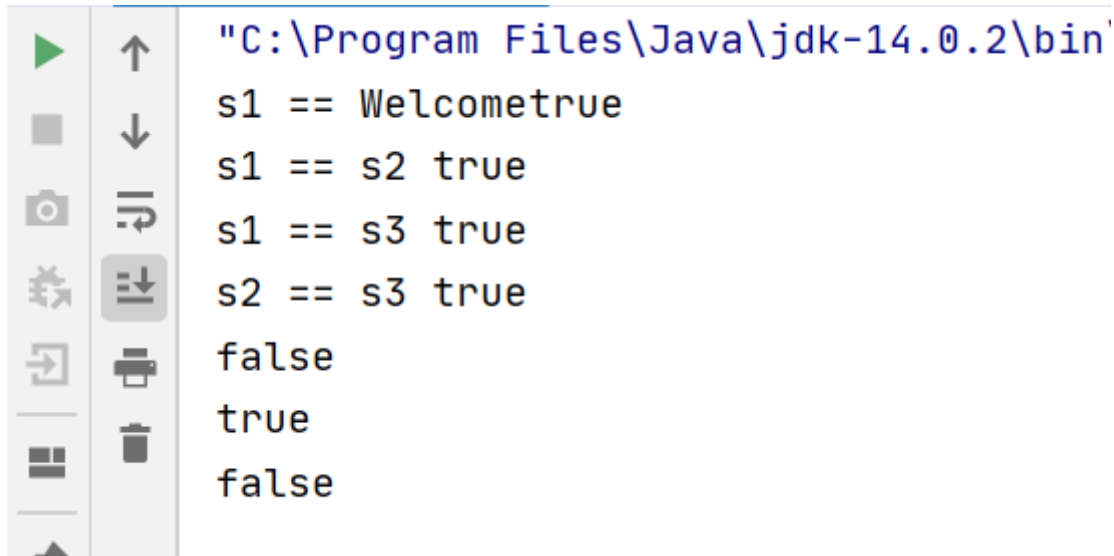
Last print statement:

The last print statement prints the summation of the values of a and b and calls upon the method foo() where the values of a and b are added first before printing as no string is printed first. The output is now "7foo".

Therefore, we get the output "72 7 34 foo34 7foo" and thus by the light of the above discussion, we can explain the reasoning behind such output.

Q2. v. Compiling the given java programs and generating the Output of the given code segments and stating the reasons:

Output:



```
"C:\Program Files\Java\jdk-14.0.2\bin'  
s1 == Welcometrue  
s1 == s2 true  
s1 == s3 true  
s2 == s3 true  
false  
true  
false
```

Fig 06: Snapshot of the output generated by the given code.

Reason:

According to the code given,

First print statement:

String s1 is a string literal. The first print statement prints "s1 == Welcome" followed by "true" since when s1.equals("Welcome") function is used, true is obtained.

Second print statement:

The string s1 and s2 are the same even though s1 is a string literal. And String s2 is a string object. So the 2nd print statement prints “s1 == s2” followed by “true” because true is returned by `s1.equals(s2)` as s1 and s2 are equal.

Third print statement:

The string s1 and s3 are string literals. Meaning, s1 and s3 indicate the same string instance. So the third print statement prints “s1 ==s3” followed by “true” because true is returned by `s1.equals(s3)` as s1 and s3 are equal.

Fourth print statement:

Prints statement prints “s2 ==s3” followed by “true” because true is returned by `s2.equals(s3)` as s2 and s3 are equal.

Prints the given string and then compares s1 with s3 using `s1.equals(s3)` which returns true as the strings are equal and hence, true is printed as well.

Fifth print statement:

In this statement, the “ == ” operator is used for comparing the strings, which is why get different results. ‘==’ operator uses reference for comparing strings instead of the actual strings. That’s why we get “false” when we compare s1 and s2, as s1 is a string literal while on the other hand, s2 is a string object.

Sixth print statement:

In this statement, the “ == ” operator is used for comparing the strings, which is why get different results. ‘==’ operator uses reference for comparing strings instead of the actual strings. That’s why we get “true” when we compare s1 and s3, as both s1 and s3 are string literals.

Seventh print statement:

In this statement, the “ == ” operator is used for comparing the strings, which is why get different results. ‘==’ operator uses reference for comparing strings instead of the actual strings. That’s why we get “false” when we compare s2 and s3, as s2 is a string object while on the other hand, s3 is a string literal.
