Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        String str = "Good"; //Line 3
        change(str); //Line 4
        System.out.println(str); //Line 5
    }

    private static void change(String s) {
        s.concat("_Morning"); //Line 9
    }
}
```

A. Good
B. _Morning
C. Good_Morning
D. None of the above

Answer: A

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Good"); //Line 3
        change(sb); //Line 4
        System.out.println(sb); //Line 5
    }

    private static void change(StringBuilder s) {
        s.append("_Morning"); //Line 9
    }
}
```

A. Good
B. _Morning
C. Good_Morning
D. None of the above

Answer: C

Q>
What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        String str1 = new String("Core");
        String str2 = new String("CoRe");
        System.out.println(str1 = str2);
    }
}
```

A. true
B. false
C. Core
D. CoRe

Answer: D

Q>
What will be the result of compiling and executing Test class?

```
public class Test extends String {
    @Override
    public String toString() {
        return "TEST";
    }

    public static void main(String[] args) {
        Test obj = new Test();
        System.out.println(obj);
    }
}
```

A. TEST
B. Output contains some string @ symbol
C. Excpetion is thrown at runtime
D. Compilation Error

Answer: D

Q>
Consider below code:

```
//Test.java
public class Test {
    public static void main(String[] args) {
        String s1 = "OCAJP";
        String s2 = "OCAJP" + "";
        System.out.println(s1 == s2);
    }
}
```

What will be the result of compiling and executing Test class?
A. OCAJP
B. true
C. false
D. CompilationError

Answer: B

Q>
Consider below code:

```
//Test.java
public class Test {
    public static void main(String[] args) {
        final String fName = "James";
        String lName = "Gosling";
        String name1 = fName + lName;
        String name2 = fName + "Gosling"; //"James Gosling"
        String name3 = "James" + "Gosling";
        System.out.println(name1 == name2);
        System.out.println(name2 == name3);
    }
}
```

What will be the result of compiling and executing Test class?

```
A. true
   true
B. true
   false
C. false
   false
D. false
   true
```

```
Answer: D

Q>
Consider below code:
//Test.java
public class Test {
    public static void main(String[] args) {
        final int i1 = 1;
        final Integer i2 = 1;
        final String s1 = ":ONE";

        String str1 = i1 + s1;//1:ONE(SCP)
        String str2 = i2 + s1;//1:ONE(HeapArea)

        System.out.println(str1 == "1:ONE");
        System.out.println(str2 == "1:ONE");
    }
}
What will be the result of compiling and executing Test class?
A. true
   true
B. true
   false
C. false
   false
D. false
   true

Answer: B

Q>
Consider below code:
//Test.java
public class Test {
    public static void main(String[] args) {
        String javaworld = "JavaWorld";
        String java = "Java";
        String world = "World";
        java += world; // java =java + world ("JavaWorld")
        System.out.println(java == javaworld);
    }
}
What will be the result of compiling and executing Test class?
A. JavaWorld
B. Java
C. World
D. true
E. false

Answer: E

Q>
Consider below code:
//Test.java
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder(100);//capacity= 100,length = 0
        System.out.println(sb.length() + ":" + sb.toString().length());
    }
}
```
What will be the result of compiling and executing Test class?
A. 100:100
B. 100:0
C. 16:16
D. 16:0
E. 0:0

Answer: E

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Hurrah! I Passed...");
        sb.delete(0, 100);
        System.out.println(sb.length());
    }
}
```
A. 19
B. 0
C. 16
D. StringIndexOutOfBoundsException

Answer: B

Q>
What will be the result of compiling and executing Test class?
```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder(5);
        sb.append("0123456789");
        sb.delete(8, 1000);
        System.out.println(sb);
    }
}
```
A. CompilationError
B. An Exception is thrown at Runtime
C. 01234567
D. 89

Answer: C

Note:
```
String        ===> No frequent change in the data then we need to go for String.
StringBuilder ===> If the data changes frequently and if we don't want threadsafety
then we go for StringBuilder
StringBuffer  ===> If the data changes frequently and if we want threadsafety then
we go for StringBuffer


Normal class :: .java =====> compile ====> .class(HDD) ===> loader =====> JRE ===>
JVM (.class)
Proxy  class ::  =========RAM====> JRE =====> generation of .class file ===>
JVM(.class)
```

.jar ===> collection of .class files which is zipped and given to the java program as an "API".

```
+++++++++++++++++++++++++++++++++++++++++++
Object Oriented Programming Concepts(OOp's)
+++++++++++++++++++++++++++++++++++++++++++
Agenda
 a. Datahiding
 b. Abstraction
 c. Encapuslation
 d. IS-A relationship
      1. Multiple inheritance
      2. Cyclic inheritance
 e. HAs-A relationship
      a. Composition
      b. Aggregation
 f. MethodSignature
 g. Polymorphism
      1. Overloading
      2. Overriding
            a. Rules of Overriding
      3. MethodHiding

 h. Static control flow
      1. static control flow from parent to child relationship
      2. static block
 i. Instance control flow
      1. instance control flow from parent to child relationship

 j. Constructor
      1. Constructor vs instance block
        2. Rules for writing a constructor
        3. Default constructor
      4. prototype of default constructor
      5. super() vs this()
      6. Overloaded constructor
      7. Recursive functions

 k. Coupling
 l. Cohesion
 m. Object type casting
      a. Compile time checking
      b. RunTime checking
 n. Difference between ArrayList al =new ArrayList() vs List l = new ArrayList()
 o. Different ways of creating an object
 p. Singleton class.
 q. Factory method.

++++++++++
Datahiding
++++++++++
 => Outsider should never access the data directly without proper authentication
refers to "Datahiding".
 => To implement datahiding we need to use an access modifier called "private".

+++++++++++
Abstraction
+++++++++++
 => Exposing the set of services,but hiding its internal implementation details is
refered as "Abstraction".
 => To implement abstraction in java, we use "abstract class and interface".
```

```
+++++++++++++
Encapsulation
+++++++++++++
 => Binding the associated datamembers and its method into single unit is called
"Encapsulation".
 => To implement encapsulation in java we need to make sure every java class should
have the feature of
    datahiding and abstraction in it.
            encapsulation = datahiding + abstraction.

What are the core features of OOP's?
      a. inheritance =======> Reusability
      b. polymorphism ======> 1:M(behaviour)=========> Flexibility
      c. encapsulation(datahiding + abstraction) ====> security

In encapsulation,if we code private data members and public setters and getters.
Then we cant access data members from outside but we can access data members
through getters.
Then how it provide security?

  GUI
   |
   |
 CheckBalance


  GUI
   |
   |
 depositMoney(5000----> 500 *10)

public class Account
{
      private double balance;

      public double getBalance()
      {
            // validate the user by asking the credential[JDBC code]
            return balance;

      }

      public void setBalance(double balance){
            // validate the user by asking the credential[JDBC code]
            this.balance +=balance;
      }
 }

+++++++++++
Inheritance
+++++++++++
 => Establishing the relationship b/w 2 classes is refered as "inheritance".
 => It refers to "IS-A" relationship.
 => In java to promote IS-A relationship we use "extends" keyword.
 => Advantage of IS-A relationship is "re-usablity".

class Parent{
      public void methodOne(){}
}
class Child extends Parent{
      public void methodTwo(){}
}

class Test
{
      public static void main(String[] args)
      {
            Parent p =new Parent();
            p.methodOne();
            p.methodTwo();//CE

            Child c  =new Child();
            c.methodOne();

            Parent p1 =new Child();
            p1.methodOne();
            p1.methodTwo();//CE

            Child c1= new Parent();//CE

      }
}
```

Note:
Whatever the parent has by default it will be available to the child,but whatever the child has by default it won't be available to
the parent,so using parent reference we can make a call only to parent class methods but not the child class methods.
If we have child object and using the child class object we can make a call to parent class methods as well as child class
methods also.
Child class object can be collected by parent class reference,but by using parent class reference we can make a call to only
parent class methods,but not the child class methods.
Child class reference cannot be used to hold "parent class" object.

What are the types of inheritance supported in java?
 a. single level(supported)
 b. multi-level(supported)
 c. multiple(not supported through classes but through interface it is supported)

Why java won't suport multiple inheritance through class and how it supports through interface?

eg#1

```
class Parent1
{
      public void methodOne(){}
}
class Parent2
{
      public void methodOne(){}
}
class Child extends Parent1,Parent2
{

}
```

```
class Test
{
      public static void main(String[] args)
      {
            Child c= new Child();
            c.methodOne();//CE:ambigous call
      }
}

eg#1
interface Inter1
{
      public void methodOne();
}
interface Inter2
{
      public void methodOne();
}
interface Inter3 extends Inter1,Inter2
{
      public void methodOne();
}
class Child implements Inter3
{
      @Override
      public void methodOne(){

      }
}
class Test
{
      public static void main(String[] args)
      {
            Child c= new Child();
            c.methodOne();//output
      }
}
```

How many types of variables are avaialble in java langauges?
 Variables are divided based on 2 phases
      a. based on the type of value the variable holds
            1. primitive type[holds direct literal]
            2. reference type[holds the address of an object]
      b. based on the position of declaration of a variable
            1. local variable[method -> stack -> no default value]
            2. instance variable[in class -> heap -> default value based on datatype]
            3. static variable[in class -> heap -> default value based on datatype]

```
instance variable -> Memory will given during the creation of an object.
              -> Memory will be deallocated once the object is destroyed.

static variable  -> Memory will be given at the time of loading the .class file.
             -> Memory will be deallocated at the time of unloading the .class file.

local variable   -> Memory will be given once the control enters inside the method.
             -> Memory will be deallocated once the control leaves the method.
```

```
eg#1.
public class Sample
{
      int i;
      public static void main(String[] args)
      {
            System.out.println(i);//CE
            Sample t=new Sample();
            System.out.println(t.i);
            t.methodOne();
      }
      public void methodOne()
      {
            System.out.println(i);
      }
}

eg#2.
public class Sample
{
      boolean b;
      public static void main(String[] args)
      {
            System.out.println(new Sample().b);
      }
}

eg#3.
public class Sample
{
      int[] a;//arrays ----> Objects[null]
      public static void main(String[] args)
      {
            System.out.println(new Sample().a);//null
      }
}

eg#3.
public class Sample
{
      int[] a;
      public static void main(String[] args)
      {
            Sample s= new Sample();
            s.a = new int[5];
            System.out.println(s.a.length);
      }
}


Q>
public class Batman {
   int squares = 81;
   public static void main(String[] args) {
     new Batman().go();
   }
   void go() {
       incr(++squares);
      System.out.println(squares);
   }
   void incr(int squares) { squares += 10; }
}
```

What is the result?
A. 81
B. 82
C. 91
D. 92
E. Compilation fails.
F. An exception is thrown at runtime.

Answer: B

Q>
Given:

```
  public class Pass {
    public static void main(String [] args) {
      int x = 5; //local variable
      Pass p = new Pass();
      p.doStuff(x);
      System.out.print(" main x = " + x);
   }
   void doStuff(int x) {
      System.out.print(" doStuff x = " + x++); // x  = 5 (used) later = 6
   }
 }
```

What is the result?
A. Compilation fails.
B. An exception is thrown at runtime.
C. doStuff x = 6 main x = 6
D. doStuff x = 5 main x = 5
E. doStuff x = 5 main x = 6
F. doStuff x = 6 main x = 5

Answer: D