Can a default method of interface override/write the public method of Object class with same signature and return type.

```
interface Alpha                                   class Object
{                                                 {
   public default String toSting()                   public String toString()
   {                                                 {
   }                                                 }
}                                                 }
              class Implementer extends Object implements Alpha
              {
                 //Ambiguity issue
              }
```

* No we **cannot write Object class public methods inside an interface as a default because we have following two problems :**

1) Ambiguity Issue :
----
default method of an interface and Object class method both will be available to the implementer class (as shown in the above program) so, We have ambiguity issue.

2) Priority :
----
* In between default and concrete method, concrete method is having more priority so, always Object class concrete method will be executed.

Due to above two reasons, Java compiler is not allowing to wtite Object class methods as a default method inside an interface.

```java
package com.ravi.functional_interface;

interface Alpha
{
   default String toString() //error not allowed
   {
      return null;
   }
}
public class IntereFaceMember
{
   public static void main(String[] args)
   {

   }
}
```

Interface from JAVA 9V

We can write private static and private non static (not public) methods inside an interface from java 9 version.

The main purpose of providing these two methods inside an interface are as follows :

1) Code Reusability

If two or more than two default methods want to share a common code (Helper Method code) then we can write these common code in private methods so it will enhance code reusability.

2) Hide the Logic from Outer World

By writing these code in private static and private non static methods the actual logic is not visible to the outer world so, It is way to make our interface as a fully abstract class so 100% abstraction is possible.

Note : By default interface is not Fully abstract but we can make it full abstract from java 9V by writing the logic inside private method.

Note : from default method we can call private static as well as private non static methods but from public static method (JDK 1.8) of interface we can call only private static method (Java 9v).

```java
package com.ravi.functional_interface;

interface Member
{
   int A = 90;  //JDK 1.0 onwards

   void m1(); //JDK 1.0 onwards


   default void m2() //JDK 1.8 onwards
   {
      m4();
      m5();
   }

   static void m3() //JDK 1.8 onwards
   {
      m5();
   }

   private void m4() //Java 9 onwards
   {
      System.out.println("Private non static method");
   }

   private static void m5() //Java 9 onwards
   {
      System.out.println("Private static method");
   }


   default void m6()
   {
      m4(); m5();
   }
}

class Implementer implements Member
{
   @Override
   public void m1()
   {
      System.out.println("Overridden abstract method");
   }
}
public class InterfaceMemberFromJava9
{
   public static void main(String[] args)
   {
      Member i = new Implementer();
      i.m1();
      i.m2();

      Member.m3();
   }
}
```

What is Marker interface in Java ?
----
* If an interface does not contain any static field and methods, Basically It is an empty interface is called Marker interface. [Empty OR Tag interface]

Example :

```
public interface Drawable  //Marker interface
{

}
```

* In java, We have following predefined marker interfaces :

1) java.lang.Cloneable
2) java.io.Serializable
3) java.util.RandomAccess

* THE MAIN PURPOSE OF MARKER INTERFACE TO PROVIDE ADDITIONAL INFORMATION TO THE JVM REGARDING THE OBJECT LIKE Object is Cloneable OR Serializable OR Randonmly Accessible

---

****What is difference between abstract class and interface ?

The following are the differences between abstract class and interface.

1) An abstract class can contain instance variables but interface variables are by default public , static and final (no instance variable).

2) An abstract class can have state (properties) of an object but interface can't have state of an object.

3) An abstract class can contain constructor but inside an interface we can't define constructor.

4) An abstract class can contain instance and static blocks but inside an interface we can't define any blocks.

5) Abstract class can't refer Lambda expression but using Functional interface we can refer Lambda Expression.

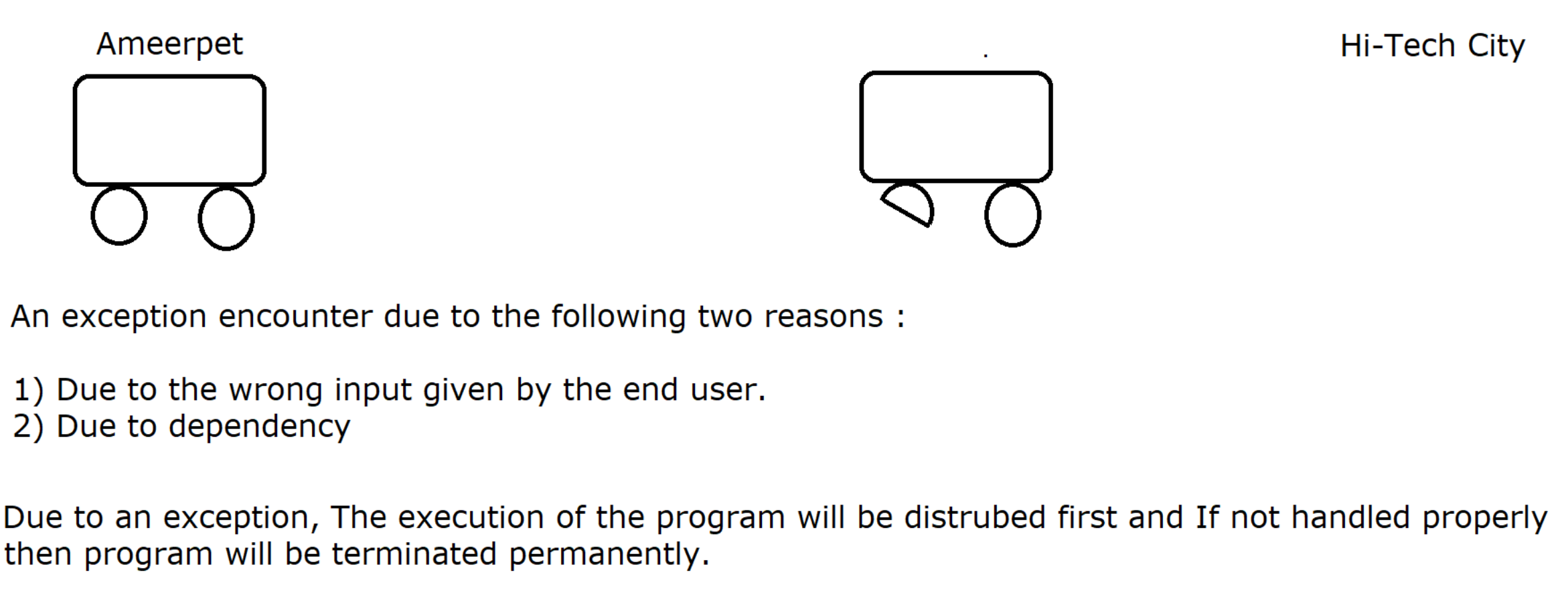
6) By using abstract class multiple inheritance is not possible but by using interface we can achieve multiple inheritance.

--------------OOPs Completed..

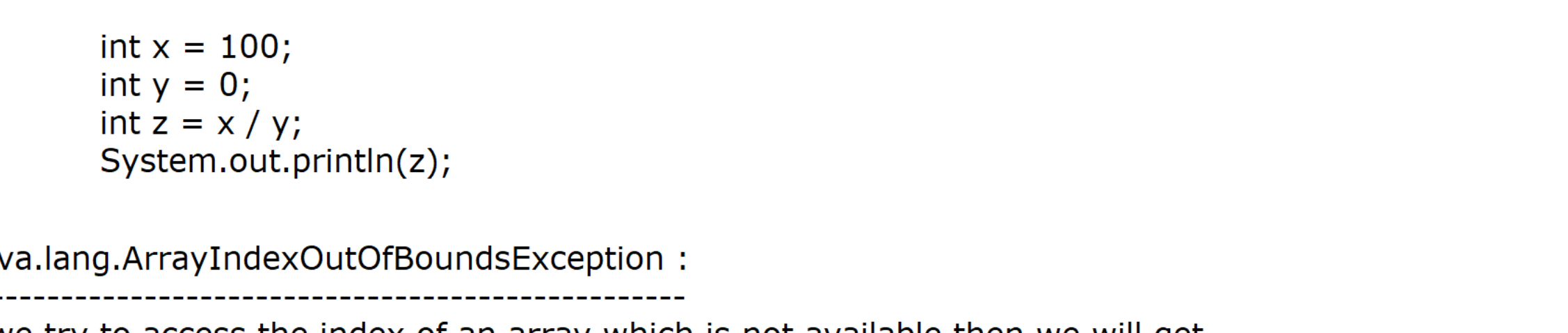
Remaining Topics :
----
1) Offline
   a) Exception Handling (6 sessionss)
   b) Array and String Logical (5 sessions)
   c) Multithreading (9 Sessions)
   d) Collection Framework (21 sessions)
   e) Stream API + Java new Features (7 - 8 Session)

2) Online (Sunday)
   a) Inner class (2 sessions)
   b) Enum in java (2 sessions)
   c) Input and Output + File Handling (2 sessions)

Exception Handling :
----
* An exception is a Runtime Error which always encounter at **Runtime only.**



What is an Exception ?
----
* An exception (which is a Runtime error) is an **abnormal situation** OR **un-expected** Situation in a normal execution flow.



* An exception encounter due to the following two reasons :

1) Due to the wrong input given by the end user.
2) Due to dependency

* Due to an exception, The execution of the program will be distrubed first and If not handled properly then program will be terminated permanently.

Different Criteria for Exception generation :
----
1) If we divide a number by zero (an int value) then we will get an exception i.e java.lang.ArithmeticException

```java
int x = 100;
int y = 0;
int z = x / y;
System.out.println(z);
```

2) java.lang.ArrayIndexOutOfBoundsException :
----
* If we try to access the index of an array which is not available then we will get java.lang.ArrayIndexOutOfBoundsException.

```java
int []arr = {10, 20, 30};
System.out.println(arr[3]);
```