System.exit(0)

- This argument acts as status code, Instead of Zero, we can't take any integer value
- Zero means normal termination, non zero means abnormal termination
- This status code internally used by JVM,whether it is zero or non-zero there is no change in the result and effect is the same w.r.t program

Difference b/w final,finally and finalize

final
- final is the modifier applicable for classes, methods and variables
- If a class is declared as the final then child class creation is not possible.
- If a method is declared as the final then overriding of that method is not possible.
- If a variable is declared as the final then reassignment is not possible.

finally
- It is a final block associated with try-catch to maintain clean up code, which should be executed always irrespective of whether exceptions are raised or not raised and whether handled or not handled.

finalize
- It is a method, always invoked by Garbage Collector just before destroying an object to perform cleanup activities.

Note
- finally block meant for cleanup activities related to try block whereas finalize() method for cleanup activities related to object.
- To maintain cleanup code finally block is recommended over finalize() method because we can't expect exact behaviour of GC.

Handling vs Ducking an Exception
- It is highly recommended to handle exceptions
- In our program the code which may rise exception is called "risky code"
- We have to place our risky code inside the try block and corresponding handling code inside the catch block.

Syntax

```
try{
     ...
     ... risky code
     ...
 }catch(XXXX e){
    ...
    ... handling code
    ...
}
```

Code without using try catch

```
class Test{
 public static void main(String... args){
  System.out.println("statement1");
  System.out.println(10/0);
  System.out.println("statement2");
 }
}
```

Output:
statement1
RE: AE:/by zero at Test.main()
Abnormal termination

with using try catch

```
public class Test{
 public static void main(String... args){
  System.out.println("statement1");
  try{
      System.out.println(10/0);
  }catch(ArithmeticException e){
      System.out.println(10/2);
  }
  System.out.println("statement2");
 }
}
```

Output:
Statement1
5

Statement2
Rethrowing an Exception(throw, throws) and Custom Exception

throw keyword in java

- This keyword is used in java to throw the exception object manually and inform jvm to handle the exception.

Syntax:: throw new ArithmeticException("/ by zero");

Eg#1.

```
class Test{
 public static void main(String... args){
  System.out.println(10/0);
 }
}
```

- Here the jvm will generate an Exception called "ArithmeticException", since main() is not handling it will handover the control to jvm, jvm will handover to DEH to dump the exception object details through printStackTrace().

vs

```
class Test{
 public static void main(String... args){
  throw new ArithmeticException("/by Zero");
 }
}
```

- Here the programmer will generate ArithmeticException,and this exception object will be delegated to JVM, jvm will handover the control to DefaultExceptionHandler to dump the exception information details through printStackTrace().
- throw keyword is mainly used to throw an customised exception not for predefined exception.

Eg#2

```
class Test{
 static ArithmeticException e =new ArithmeticException();
 public static void main(String... args){
  throw e;
 }
}
```

Output:
Exception in thread "main" java.lang.ArithmeticException

Eg#3

```
class Test{
 static ArithmeticException e;
 public static void main(String... args){
  throw e;
 }
}
```

Output:
Exception in thread "main" java.lang.NullPointerException.

Case2
- After throw statement we can't take any statement directly otherwise we will get a compile time error saying an unreachable statement.

Eg#1

```
class Test{
 public static void main(String... args){
  System.out.println(10/0);
  System.out.println("hello");
 }
}
```

Output:
Exception in thread "main" java.lang.ArithmeticException

Eg#2

```
class Test{
  public static void main(String... args){
  throw new ArithmeticException("/ by zero");
  System.out.println("hello");
  }
}
```

Output:
CompileTime error
Unreachable statement
System.out.println("hello");

Case3

- We can use throw keyword only for Throwable types otherwise we will get a compile time error saying incompatible type.

Eg#1

```
class Test3{
 public static void main(String... args){
  throw new Test3();
 }
}
```

Output:
Compile time error.
found::Test3
required:: java.lang.Throwable

Eg#2

```
public class Test3 extends RunTimeException{
 public static void main(String... args){
  throw new Test3();
 }
}
```

Output:
RunTimeError: Exception in thread "main" Test3

Customized Exceptions (User defined Exceptions)

- Sometimes we can create our own exception to meet our programming requirements.
- Such type of exceptions are called customised exceptions (user defined exceptions).