linkedin id :: https://in.linkedin.com/in/nitin-m-110169136

++++++++++++++++++++++++++++++++
Loose coupling and tight coupling
++++++++++++++++++++++++++++++++

```
Parent  p = new Parent();
  p. xxxxxxx(); // object is tightly coupled.

Parent  p = new Child();
   p. xxxxxxx();//object is loosely coupled.


List l = new ArrayList();
       new LinkedList();
       new Stack();
       new Vector();
```

++++++++++++++++++++++++
   Exception Handling
++++++++++++++++++++++++

throw vs throws

throw -> Sometimes we need to create an exception object explicitly and we can hand over the exception object to JVM
manually, to do this we use throw keyword in java.

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

case1::

```
class Test
{
      public static void main(String... args)
      {
            //Cretion of Arithmetic Exception object and handing over to the jvm will be done automaticaly by main()
            System.out.println(10/0);
      }
}
```

case2::

```
class Test
{
      public static void main(String... args)
      {
            //We are creating an ArithmeticException object explicitly and handing over to JVM manually.
            throw new ArithmeticException("/by zero");
      }
}
```

Note: we use throw keyword normally for customized expections not for pre-defined exceptions.

case1:
eg::

```
class Test
{
```

```
	static ArithmeticException e = new ArithmeticException();
	public static void main(String[] args)
	{
		throw e;//Runtime exception : Exception in thread "main"
java.lang.ArithmeticException
	}
}

eg::
class Test
{
	static ArithmeticException e; //e  =null (default value given by JVM)
	public static void main(String[] args)
	{
		throw e;//Runtime exception : Exception in thread "main"
java.lang.NullPointerException
	}
}

case 2::
class Test
{
	public static void main(String[] args)
	{
		System.out.println(10/0);
		System.out.println("hello");//Runtime exception : Exception in thread
"main" java.lang.ArithmeticException
	}
}
class Test
{
	public static void main(String[] args)
	{
		throw new ArithmeticException("/by zero");
		System.out.println("hello");//CE: unreachable statement
	}
}

case3::
class Test
{
	public static void main(String[] args)
	{
		throw new Test(); //CE: incompatible types
	}
}
class Test extends RuntimeException
{
	public static void main(String[] args)
	{
		throw new Test();//RuntimeException : exception in thread "main" Test
	}
}

+++++++
throws
+++++++
eg#1.
class Test
```

```
{
	public static void main(String[] args)
	{
		PrintWriter out = new PrintWriter("abc.txt");
		out.println("hello");
	}
}
```
CE: unreported exception java.io.FileNotFoundException; must be caught or declared to be thrown

eg#2
```
class Test
{
	public static void main(String[] args)
	{
		Thread.sleep(1000);
	}
}
```
CE: unreported exception java.lang.InterruptedException; must be caught or declared to be thrown

We can handle the compile time error in 2 ways
	a. using try catch(handles both checked and unchecked exception)
	b. using throws keyword(meant for handling checked exception only)

eg#3.
```
class Test
{
	public static void main(String[] args)
	{
		try{
			Thread.sleep(1000);
		}catch(InterruptedException e){
			e.printStackTrace();
		}
	}
}
```

eg#4
```
class Test
{
	public static void main(String[] args) throws InterruptedException
	{
		Thread.sleep(1000);
	}
}
```

Note:
1. The main responsiblity of "throws" keyword is to delegate the responsiblity of exception handling to the caller method.
2. "throws" keyword is required only for checked excpetion,usage of throws keyword is of no use for uncheckedexceptions.
3. "throws" keyword is required only to convince the compiler,at the runtime if the problem occurs then there would be
    a possiblity of abnormal termination of the program.
4. since throws keyword doesnot guarantee the smoothful termination of the program, it is best suggested to use
    "try with catch block".

```
eg#1.
class Test{

	public static void main(String... args) throws InterruptedException{
		doStuff();
	}
	public static void doStuff() throws InterruptedException{
		doMoreStuff();
	}
	public static void doMoreStuff() throws InterruptedException{
		Thread.sleep(5000);
	}
}

case1::
 class Test
 {
	public static void main(String... args)throws Test //CE: incompatible types
	{

	}
 }
 class Test extends RuntimeException
 {
	public static void main(String... args)throws Test
	{

	}
 }

case2::
class Test
{
	public static void main(String... args)
	{
		throw new Exception();//CE: unreported exception must be caught or
declared to be thrown.
	}
}
class Test
{
	public static void main(String... args)
	{
		throw new Error();//Runtime error: Exception in thread "main"
java.lang.Error.
	}
}

case3::
class Test
{
	public static void main(String... args)
	{
		try{
			System.out.println("hello");//hello
		}catch(Exception e){//partially checked so no problem

		}
	}
```

```
}
class Test
{
	public static void main(String... args)
	{
		try{
			System.out.println("hello");//hello
		}catch(ArithmeticException e){//unchecked excpetion so no problem

		}
	}
}
class Test
{
	public static void main(String... args)
	{
		try{
			System.out.println("hello");
		}catch(java.lang.IOException e){
			//CE: exception java.io.IOException is never thrown in the body
of corresponding try statement
		}
	}
}
class Test
{
	public static void main(String... args)
	{
		try{
			System.out.println("hello");
		}catch(java.lang.InterruptedException e){
		   //CE: exception java.io.InterruptedException is never thrown in the
body of corresponding try statement
		}
	}
}
class Test
{
	public static void main(String... args)
	{
		try{
			System.out.println("hello");//hello
		}catch(Error e){//unchecked exception so no problem

		}
	}
}

case4::
 class Test throws Exception //invalid
 {
	Test() throws Exception{//valid

	}
	public void m1()throws Exception{//valid

	}
 }
```

```
Note:
try{
   int a = 10/0;
}catch(Exception e){
      e.printStackTrace();
}catch(ArithmeticException e){//CE: Exception java.lang.ArithmeticException has
already been caught
      e.printStackTrace();
}

2>
try{
      //risky code
}catch(ArithmeticException e){
      //handling code
}catch(NullPointerException e){
      //handling code
}catch(ClassCastException e){
      //handling code
}catch(Exception e){
      //handling code
}finally{
      //to maintain cleanup code
}

throw  :: to handover the create exception object to jvm manually(prefered for
custom exceptions).
throws :: to delegate the responsiblity of handling the exception object to the
caller method.

++++++++++++++++++++
ClassNotFoundException
++++++++++++++++++++
try{
  Class.forName("com.mysql.cj.jdbc.Driver");
   //JVM search in  a. bootstrap class loader(rt.jar),
  // b. extension classloader(jre/ext)
 //  c. application class loader(classpath)
}catch(ClassNotFoundException e){
      e.printStackTrace();
}

NoClassDefFoundError
      A.java <---- B.java

try{
      Class.forName("A");
}catch(ClassNotFoundException e){
      e.printStackTrace();
}catch(NoClassDefFoundError e){
      e.printStackTrace();
}

++++++++++++++++++
ClassCastException
++++++++++++++++++
class Test{
      public static void main(String... args)
      {
```

```
            String s   = new String("sachin");
            Object obj = (Object)s;//valid
      }
}
class Test{
      public static void main(String... args)
      {
            Object obj = new Object();
            String s = (String)obj;//RuntimeException  :
java.lang.ClassCastException
      }
}
class Test{
      public static void main(String... args)
      {
            Object obj = new String("sachin");
            String s = (String)obj;//valid
      }
}

++++++++++++++++++++++++++
ExceptionInInitializerError
++++++++++++++++++++++++++
 => It is a child class of RuntimeException and it is unchecked.
 => This exception would be raised automatically when any exception occurs  while
performing static variable initialization
     or static block execution.
class Test{
      static int i = 10/0; // RuntimeException :
java.lang.ExceptionInInitializerError
}

class Test{
      static{
            String s= null;
            System.out.println(s.length());// RuntimeException :
java.lang.ExceptionInInitializerError
      }
}

++++++++++++++++++++++
IllegalArgumentException
++++++++++++++++++++++
class Test{
      public static void main(String... args)
      {
            Thread t = new Thread();//[1...10]
            t.setPriority(10);//valid
            t.setPriority(100);//invalid[RuntimeException :
java.lang.IllegalArgumentException]
      }
}

++++++++++++++++++++
NumberFormatException
++++++++++++++++++++
class Test{
      public static void main(String... args)
      {
```

```
            Integer i = Integer.parseInt("10");
            Integer j = Integer.parseInt("Ten");//NumberFormatException
      }
}
```

Given:

```
public class Yikes {
 public static void go(Long n) {
      System.out.print("Long ");
 }
 public static void go(Short n) {
      System.out.print("Short ");
 }
 public static void go(int n) {
      System.out.print("int ");
 }
 public static void main(String[] args) {
      short y = 6; //short ---->int
      long z = 7;//long ---> float ----> double
      go(y);
      go(z);
 }
}
```

What is the result?

A. int Long
B. Short Long
C. Compilation fails.
D. An exception is thrown at runtime.

Answer: A

Given:

```
class Alpha {
      public void foo() { System.out.print("Afoo "); }
}
public class Beta extends Alpha {
      public void foo() { System.out.print("Bfoo "); }
      public static void main(String[] args) {
            Alpha a = new Beta();
            Beta b  = (Beta)a;
            a.foo();
            b.foo();
      }
}
```

What is the result?

A. Afoo Afoo
B. Afoo Bfoo
C. Bfoo Afoo
D. Bfoo Bfoo
E. Compilation fails.
F. An exception is thrown at runtime.

Answer: D

Q>
Given:

```
class Animal {
      public String noise() {
            return "peep";
      }
```

```
}
class Dog extends Animal {
      public String noise() {
            return "bark";
      }
}
class Cat extends Animal {
      public String noise() {
            return "meow";
      }
}
...
30. Animal animal = new Dog();
31. Cat cat = (Cat)animal;
32. System.out.println(cat.noise());
```

What is the result?
A. peep
B. bark
C. meow
D. Compilation fails.
E. An exception is thrown at runtime.

Answer: E

Given:

```
1. interface DeclareStuff {
2.    public static final int EASY = 3;
3.
4.    void doStuff(int t);
5. }
6.
7. public class TestDeclare implements DeclareStuff {
8.          public static void main(String[] args) {
9.                int x = 5;
10.               new TestDeclare().doStuff(++x);
11.         }
12.
13.         void doStuff(int s) {
14.               s += EASY + ++s;
15.               System.out.println("s " + s);
16.         }
17.}
```

What is the result?
A. s 14
B. s 16
C. s 10
D. Compilation fails.
E. An exception is thrown at runtime.

Answer: D

Q>

```
interface DoStuff2 {
      float getRange(int low, int high);
}
interface DoMore {
      float getAvg(int a, int b, int c);
}
```

```
abstract class DoAbstract implements DoStuff2, DoMore {

}
class DoStuff implements DoStuff2 {
      public float getRange(int x, int y) {
            return 3.14f;
      }
}
interface DoAll extends DoMore {
      float getAvg(int a, int b, int c, int d);
}
```

What is the result?

A. The file will compile without error.
B. Compilation fails. Only line 7 contains an error.
C. Compilation fails. Only line 12 contains an error.
D. Compilation fails. Only line 13 contains an error.
E. Compilation fails. Only lines 7 and 12 contain errors.
F. Compilation fails. Only lines 7 and 13 contain errors.
G. Compilation fails. Lines 7, 12, and 13 contain errors.

Answer: A

```
++++++++++++++++++++
  MultiThreading
++++++++++++++++++++
 1. Need of MultiThreading
 2. Different types of creating a Thread
      a. extends Thread class
      b. implements Runnable interface
 3. Thread class constructors
 4. Thread priority
 5. Getting and Setting name for a Thread
 6. How to prevents Threads from Execution
      a. yield()
      b. join()
      c. sleep()
 7. Synchronization(notify(),notifyAll(),wait())
 8. Inter Thread Communication
 9. Dead Lock
 10. Daemon Threads
 11. Conclusions of a Thread
      a. How to stop a thread
      b. Suspend and rename a thread
      c. Thread group
      d. Green Thread
      e. Thread local
 12. LifeCycle of a Thread

eg#1.
class MyThread extends Thread
{
      @Override
      public void run(){
            for(int i = 0;i<=5;i++)
                  System.out.println("child thread");
      }
}
class Sample
{
```

```
	public static void main(String... args)
	{
		MyThread t = new MyThread();
		t.start();
		// main thread[5] , thread-0[5]
		for(int i = 0;i<=5;i++)
			System.out.println("main thread");
	}
}

main() ---> JVM --create a thread by the name "main" ->  execution of main()
starts....
start() ---> main thread ----> ............

 Ouptut :: irregular output(because which threads executes can't be predicted....)


case2::
 t.start() ====> new thread will be created which is responsible for executing
run().
 t.run()   ====> no new thread will be created and run() will be called just like a
normal method by main thread.

case3::
 What is the importance of Thread class start method?

 start()
 {
	1. Register the thread with ThreadScheduler.
	2. All the other mandatory low level activities.
	3. invoke or call run() method.
 }
 without executing Thread class start(),there is no chance of creating a new thread
in java so we say start() as heart of
 "MultiThreading".


case4:
public class Thread implements Runnable
{
	public void start()
	{
		1. Register the thread with ThreadScheduler.
		2. All the other mandatory low level activities.
		3. invoke or call run() method.
	}
	@Override
	public void run(){
		//no implementation
	}
}
class MyThread extends Thread{}
class ThreadDemo{
	public static void main(String... args)
	{
		MyThread t =new MyThread();
		t.start();
	}
}
```

```
case5:
public class Thread implements Runnable
{
      public void start()
      {
            1. Register the thread with ThreadScheduler.
            2. All the other mandatory low level activities.
            3. invoke or call run() method.
      }
      @Override
      public void run(){
            //no implementation
      }
}
 class MyThread extends Thread{
      public void run(){System.out.println("no arg method");}
      public void run(int i){System.out.println("int arg method");}//specialized
method
 }
 class ThreadDemo{
      public static void main(String... args)
      {
            MyThread t =new MyThread();
            t.start();
      }
 }

case6:
public class Thread implements Runnable
{
      public void start()
      {
            1. Register the thread with ThreadScheduler.
            2. All the other mandatory low level activities.
            3. invoke or call run() method.
      }
      @Override
      public void run(){
            //no implementation
      }
}
class MyThread extends Thread{
      public void run(){System.out.println("run method");}
      public void start(){System.out.println("start method");}
}
class ThreadDemo{
      public static void main(String... args)
      {
            MyThread t =new MyThread();
            t.start();
            System.out.println("main method");
      }
 }
case 7:
class MyThread extends Thread{
      public void run(){System.out.println("run method");}
}
class ThreadDemo{
```

```
      public static void main(String... args)
      {
            MyThread t =new MyThread();
            t.start();
            ;;;;;;;;
            ;;;;;;;;
            ;;;;;;;;
            t.start();//RuntimeExeption : java.lang.IllegalThreadStateException
      }
 }
```

```
++++++++++++++++++++++++++++++++
 Collections with JDK8 features
++++++++++++++++++++++++++++++++
 1. Hierarchy of collections(List,Set,Map)
 2. Important interface in collections
 3. Difference b/w comparable and compartor
 4. Working with TreeSet objects
 5. Working with cursors on collection objects
 6. Working with Map objects
 7. Working with java.util.Properties(K,V)
 8. Working with Optional api and stream api, jodha api[Date and time api]
```