Throws keyword

If there is a possibility of our code raising a checked exception then we will get a compile time error if we don't handle it

Example 1 :

```
class abc{
public static void main(String args[]){
Printwriter pw = new Printwriter("abc.txt");
}
}
```

Above code will give compile time error saying unreported exception java.lang.FileNotFoundException must be caught or declared to be thrown

Now we can handle this exception in 2 ways

1. By using try catch

```
class abc{
public static void main(String args[]){
try{
Printwriter pw = new Printwriter("abc.txt");
} catch(FileNotFoundException e){
}
}
}
```

2. By using throws keyword

```
class abc{
public static void main(String args[]) throws
FileNotFoundException{
Printwriter pw = new Printwriter("abc.txt");
}
}
```

Throws keyword states that method main throws FileNotFoundException and it's the responsibility of the caller to handle it. Throws keyword is used to delegate responsibility of exception handling to the caller (It may be another method or jvm). Then in that case the caller method is responsible to handle that exception.

Throws keyword is required only for checked exceptions and the usage of throws keyword for unchecked exceptions has no use or impact. Throws keyword requires only to convince compiler and usage of throws keyword doesn’t prevent abnormal termination of the program.

```
class ABC{
public static void main(String args[]){
doOneThing();
}

void doOneThing(){
doOneMoreThing();
}

void doOneMoreThing(){
```

```
Printwriter pw = new Printwriter("abc.txt");
}
}
```

Compiler will throw error on doOneMoreThing()

If you add "*throws FileNotFoundException*" to doOneMoreThing() compiler will throw error for doOneThing(), because it will check if caller of doOneMoreThing() is handling it

If you add "*throws FileNotFoundException*" to doOneThing() compiler will throw an error for main(), because it will check if the caller of doOneThing() is handling it. Here the caller is the main() method in this case.

Once you add "*throws FileNotFoundException*" to main() compiler will compile

```
class ABC{
public static void main(String args[]) throws
FileNotFoundException{
doOneThing();
}

void doOneThing() throws FileNotFoundException{
doOneMoreThing();
}

void doOneMoreThing() throws FileNotFoundException{
Printwriter pw = new Printwriter("abc.txt");
}
}
```

Let's understand some rules :

Rule 1 : We can use throws keyword for methods and constructors but not for classes

Following code won't compile since using throws with class is invalid

```
class abc throws Exception{
void ABC() throws Exception{
}
public void m1() throws exception{
}
}
```

Rule 2 : We can use the throws keyword only for throwable types. If we are trying to use normal java classes then we will get compile time error saying incompatible types.

```
class ABC{
public void m1() throws ABC{
}
}
```

Compile time error : incompatible types found test required throwable

```
class ABC extends ArithmeticException{
public void m1() throws ABC{
}
}
```

Compiles because ArithmeticException is subclass of throwable and since ABC is extending it, it also becomes subclass of it.

Rule 3 : When using the throw keyword to throw a checked exception from within a method, the method must either :

Declare the throws clause followed by the exceptions thrown by the throw statements

OR

Catches the exceptions thrown by the throw statements.

```
class abc{
public static void main(String args[]){
throw new Exception();
}
}
```

Since this is Checked exception
Compile time error unreported exception java.lang.exception
Must be caught or declared to be thrown

```
class abc{
public static void main(String args[]){
throw new Error();
}
```

```
}
```

No compile time error
Runtime error :Exception in thread “main” Java.lang.error at abc.main()

Rule 4 :Throws keyword can be followed by one or more exception classes followed by comma.

Example :

```
class ABC{
public void m1() throws ArithmeticException,
NullPointerException{
}
}
```

This code is completely valid

Rule 5 : If the throw statements throw unchecked exceptions, the method is not required to declare those unchecked exceptions in its throws clause.