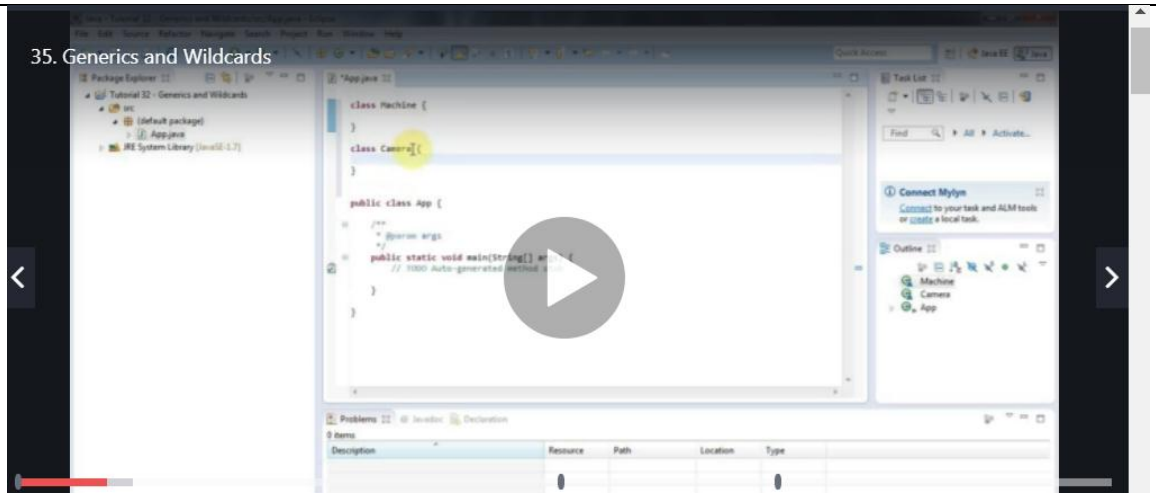
JAVA REPORT

Date:	**12/06/2020**	**Name:**	**SAFIYA BANU**
Course:	**JAVA**	**USN:**	**4AL16EC061**
Topic:	1. Generics and Wildcards 2. Anonymous Classes 3. Reading Files Using Scanner 4. Handling Exceptions 5. Multiple Exceptions 6. Runtime vs. Checked Exceptions 7. Abstract Classes 8. Reading Files With File Reader 9. Try-With-Resources 10. Creating and writing text files	**Semester & Section:**	**8TH B**
Github Repository:	**Safiya-Courses**		



1 GENERICS AND WILDCARDS

The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

Without Generics, we can store any type of objects.

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

With Generics, it is required to specify the type of object we need to store.

```
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add("10");// compile-time error
```

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

2 Anonymous Classes

Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class same time. They are like local classes except that they do not have a name. Use them if you need to use a local clas once.

This section covers the following topics:

- Declaring Anonymous Classes
- Syntax of Anonymous Classes
- Accessing Local Variables of the Enclosing Scope, and Declaring and Accessing Members of the Anonymo

- Examples of Anonymous Classes

Declaring Anonymous Classes

While local classes are class declarations, anonymous classes are expressions, which means that you define the clas another expression. The following example, HelloWorldAnonymousClasses, uses anonymous classes in the initializ statements of the local variables frenchGreeting and spanishGreeting, but uses a local class for the initialization of t variable englishGreeting:

3 READING FILES USING SCANNER

From Java 5 onwards java.util.Scanner class can be used to read file in Java. Earlier we have seen example of readin file in Java using FileInputStream and reading file line by line using BufferedInputStream and in this Java tutori we will See How can we use Scanner to read files in Java. Scanner is a utility class in java.util package and provide several convenient method to read int, long, String, double etc from source which can be an InputStream, a file a String itself. As noted on How to get input from User, Scanner is also an easy way to read user inp using System.in (InputStream) as source.Main advantage of using Scanner for reading file is that it allows you change delimiter using useDelimiter() method, So you can use any other delimiter like comma, pipe instead of whi space.

4 HANDLING EXCEPTION

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow application can be maintained.

5 MULTIPLE EXCEPTION

Each exception type that can be handled by the `catch` block is separated using a vertical bar or pipe.

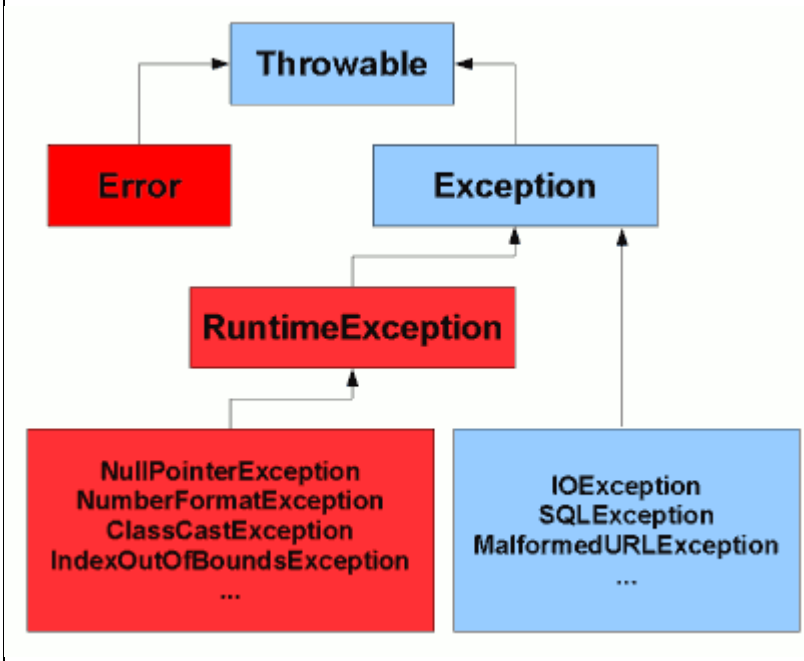
Its syntax is:

```
try {
  // code
} catch (ExceptionType1 | Exceptiontype2 ex) {
  // catch block
}
```

6 Runtime vs. Checked Exceptions

Exception Hierarchy

In Java, exceptions are broadly categorized into two sections: **checked exceptions and unchecked exceptions**.



Checked Exceptions

Java forces you to handle these error scenarios in some manner in your application code. They will come immediat
your face, once you start compiling your program. You can definitely ignore them and let them pass to JVM, but it'
habit. Ideally, you must handle these exceptions at suitable level inside your application so that you can inform the
about failure and ask him to retry/ come later.

Generally, checked exceptions denote error scenarios which are outside the immediate control of the program. They
usually interacting with outside resources/network resources e.g. database problems, network connection errors, mi
files etc.

Checked exceptions are subclasses of **Exception** class.

Example of checked exceptions are : **ClassNotFoundException**, **IOException**, **SQLException** and so on.

Checked Exception Example

FileNotFoundException is a checked exception in Java. Anytime, we want to read a file from filesystem, Java force
handle error situation where file may not be present in place.

7 ABSTRACT CLASS

Data abstraction is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either abstract classes

The abstract keyword is a non-access modifier, used for classes and methods:

- Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited fro another class).
- Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by subclass (inherited from).

8 Reading Files With File Reader

The following code shows how to read a small file using the new *Files* class:

```
1       @Test
2       public void whenReadSmallFileJava7_thenCorrect()
3         throws IOException {
4           String expected_value = "Hello, world!";
5
6           Path path = Paths.get("src/test/resources/fileTest.txt");
7
8           String read = Files.readAllLines(path).get(0);
9           assertEquals(expected_value, read);
10      }
```

9 Try-With-Resources

The try-with-resources statement is a try statement that declares one or more resources. A *resource* is an object that closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

The following example reads the first line from a file. It uses an instance of BufferedReader to read data from the file. BufferedReader is a resource that must be closed after the program is finished with it:

```
static String readFirstLineFromFile(String path) throws IOException {
   try (BufferedReader br =
                 new BufferedReader(new FileReader(path))) {
      return br.readLine();
   }
}
```

10 Creating and writing text files

1. Create file with java.io.File class

Use File.createNewFile() method to create new file. This method returns a boolean value –

- `true` if the file is created successfully.
- `false` if the file already exists or the operation failed for some reason.