Packages

- A package in java is a group of related classes and interfaces
- Packages help in categorizing and maintaining classes and interfaces
- Packages provide access protection and help in avoiding naming collision
- Package is created using package keyword at the top of the file where class is created
- Packages can be imported using import statement
- Java provides with many in-built packages like java.util,java.io,

Static members of class

- A member variable or method declared as static is shared among all objects
- Only one copy of such a member is created in memory
- A static method cannot access non static variables of class

Static members of class

- Static variables
- Static Methods
- Static class

Static Block

- A block can be declared as static for initializing static variables of class
- Such a block is called before main method

Logging API

- In Java, **logging** is an important feature that helps developers to trace out the errors. Java is the programming language that comes with the **logging** approach. It provides a **Logging API** that was introduced in Java 1.4 version. It provides the ability to capture the log file

Logging API

- **Logging** is an API that provides the ability to trace out the errors of the applications. When an application generates the logging call, the Logger records the event in the LogRecord. After that, it sends to the corresponding handlers or appenders. Before sending it to the console or file, the appenders format that log record by using the formatter or layouts.

Components of Logger

- The Java Logging components used by the developers to create logs and passes these logs to the corresponding destination in the proper format. There are the following three core components of the Java logging API:
- Loggers
- Logging Handlers or Appender
- Logging Formatters or Layouts

Loggers

- The code used by the client sends the log request to the Logger objects. These logger objects keep track of a log level that is interested in, also rejects the log requests that are below this level.

Logging Handlers

Java Logging API allows us to use multiple handlers in a Java logger and the handlers process the logs accordingly. There are the five logging handlers in Java:

- **StreamHandler:** It writes the formatted log message to an OutputStream.
- **ConsoleHandler:** It writes all the formatted log messages to the console.
- **FileHandler:** It writes the log message either to a single file or a group of rotating log files in the XML format.
- **SocketHandler:** It writes the log message to the remote TCP ports.
- **MemoryHandler:** It handles the buffer log records resides in the memory.

Logging Formatters or Layouts

- The logging formatters or layouts are used to format the log messages and convert data into log events. Java SE provides the following two standard formatters class:
- SimpleFormatter
- XMLFormatter

Java Logging Levels

FINEST	Specialized Developer Information
FINER	Detailed Developer Information
FINE	General Developer Information
CONFIG	Configuration Information
INFO	General Information
WARNING	Potential Problem
SEVERE	Represents serious failure
Special Log Levels	
OFF	Turns off the logging
ALL	Captures everything

Using Logging libraries

- Simple Logging Facade for Java (abbreviated SLF4J) acts as a facade for different logging frameworks (e.g., java.util.logging, logback, Log4j). It offers a generic API, making the logging independent of the actual implementation.
- This allows for different logging frameworks to coexist. And it helps migrate from one framework to another. Finally, apart from standardized API, it also offers some "syntactic sugar."

Maven

- Maven, a Yiddish word meaning *accumulator of knowledge*, Maven is a tool that can be used for building and managing any Java-based project. It has been created to make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

SlF4J

```
public class App
{
    public static void main( String[] args )
    {
     Logger logger = LoggerFactory.getLogger(App.class);

         logger.info("This is an information message");
         logger.error("this is a error message");
         logger.warn("this is a warning message");
    }
}
```

Cohesion

- Cohesion in Java is the Object-Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose. In object-oriented design, cohesion refers to how a single class is designed

Cohesion

- The advantage of high cohesion is that such classes are much easier to maintain and less frequently changed than classes with low cohesion. Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.

Loose Coupling

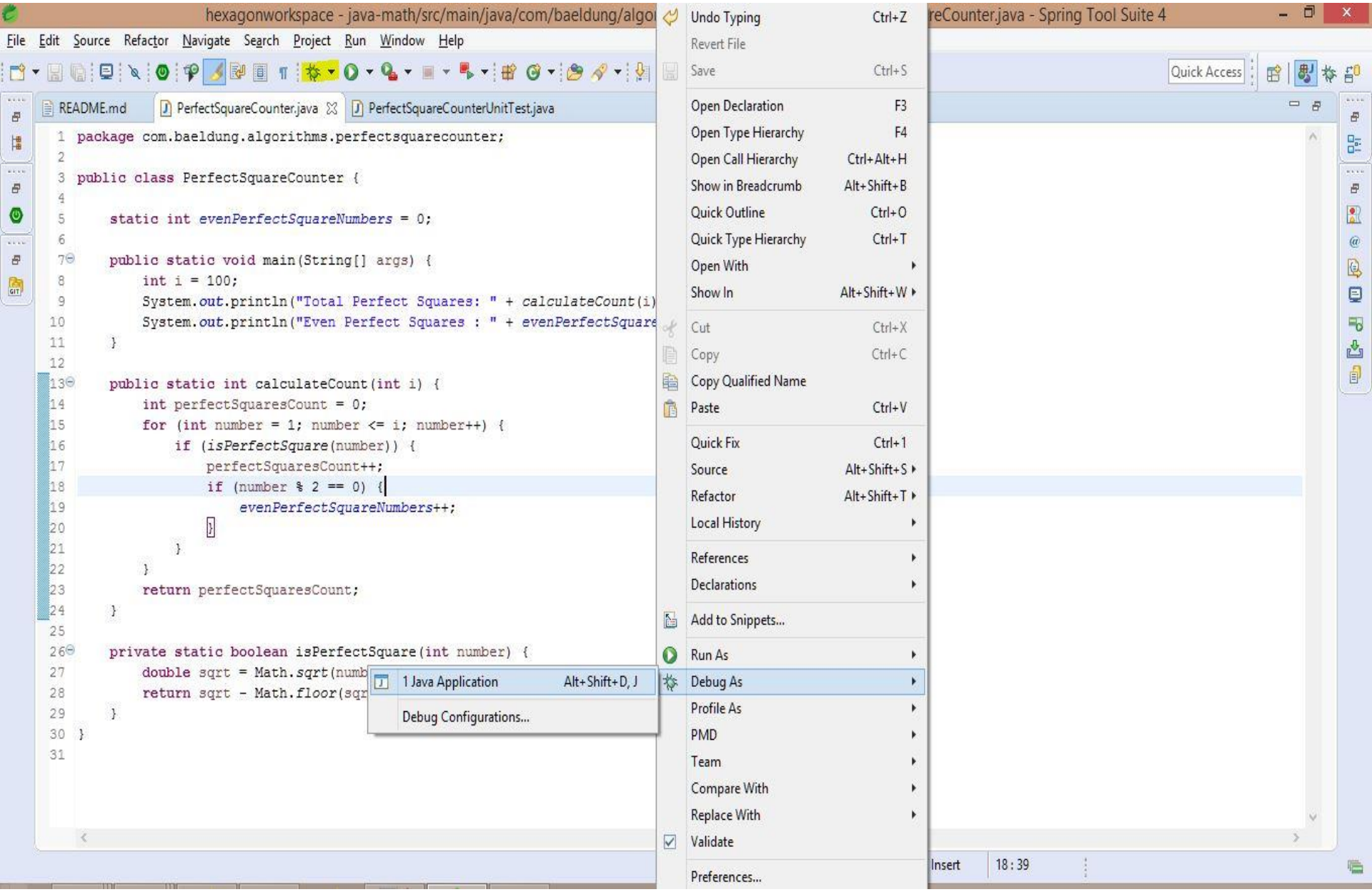
- When two classes, modules, or components have low dependencies on each other, it is called loose coupling in Java. Loose coupling in Java means that the classes are independent of each other. The only knowledge one class has about the other class is what the other class has exposed through its interfaces in loose coupling. If a situation requires objects to be used from outside, it is termed as a loose coupling situation.

Debugging a Java Application

- Eclipse has great support for debugging an application. **It visualizes step-by-step execution** and helps us uncover bugs.
- To demonstrate the debugging features in Eclipse, we'll use a sample program *PerfectSquareCounter*. This program counts the total perfect squares and even perfect squares under a given number:

```java
public class PerfectSquareCounter {

    static int evenPerfectSquareNumbers = 0;

    public static void main(String[] args) {
        int i = 100;
        System.out.println("Total Perfect Squares: " +
calculateCount(i));
        System.out.println("Even Perfect Squares : " +
evenPerfectSquareNumbers);
    }

    public static int calculateCount(int i) {
        int perfectSquaresCount = 0;
        for (int number = 1; number <= i; number++) {
            if (isPerfectSquare(number)) {
                perfectSquaresCount++;
                if (number % 2 == 0) {
                    evenPerfectSquareNumbers++;
                }
            }
        }
        return perfectSquaresCount;
    }

    private static boolean isPerfectSquare(int number) {
        double sqrt = Math.sqrt(number);
        return sqrt - Math.floor(sqrt) == 0;
    }
}
```

- First, we need to **start the Java program within Eclipse in debug mode.** This can be achieved in two ways:
- Right-click on the editor and select *Debug As -> Java Application* (shown in below screenshot)
- *Debug* the program from the toolbar (highlighted in below screenshot)


hexagonworkspace - java-math/src/main/java/com/baeldung/algo
reCounter.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
README.md
PerfectSquareCounter.java
PerfectSquareCounterUnitTest.java
package com.baeldung.algorithms.perfectsquarecounter;
public class PerfectSquareCounter {
static int evenPerfectSquareNumbers = 0;
public static void main(String[] args) {
int i = 100;
System.out.println("Total Perfect Squares: " + calculateCount(i)
System.out.println("Even Perfect Squares : " + evenPerfectSquare
}
public static int calculateCount(int i) {
int perfectSquaresCount = 0;
for (int number = 1; number <= i; number++) {
if (isPerfectSquare(number)) {
perfectSquaresCount++;
if (number % 2 == 0) {
evenPerfectSquareNumbers++;
}
}
}
return perfectSquaresCount;
}
private static boolean isPerfectSquare(int number) {
double sqrt = Math.sqrt(numb
return sqrt - Math.floor(sqr
}
}
1 Java Application Alt+Shift+D, J
Debug Configurations...
Undo Typing Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
References
Declarations
Add to Snippets...
Run As
Debug As
Profile As
PMD
Team
Compare With
Replace With
Validate
Preferences...
Insert
18 : 39

Break Points

- We need to **define the points at which the program execution should pause** for investigation. **These are called breakpoints and are applicable for methods.** They can also be defined anytime before or during execution.
- Basically, there are 3 ways to add breakpoints to the program:
- Right-click on the marker bar (vertical ruler) corresponding to the line and select Toggle Breakpoint (shown in the below screenshot)
- Press *Ctrl+Shift+B* on the necessary line while in the editor
- Double-click on the marker bar (vertical ruler) corresponding to the necessary line

README.md | PerfectSquareCounter.java | PerfectSquareCounterUnitTest.java

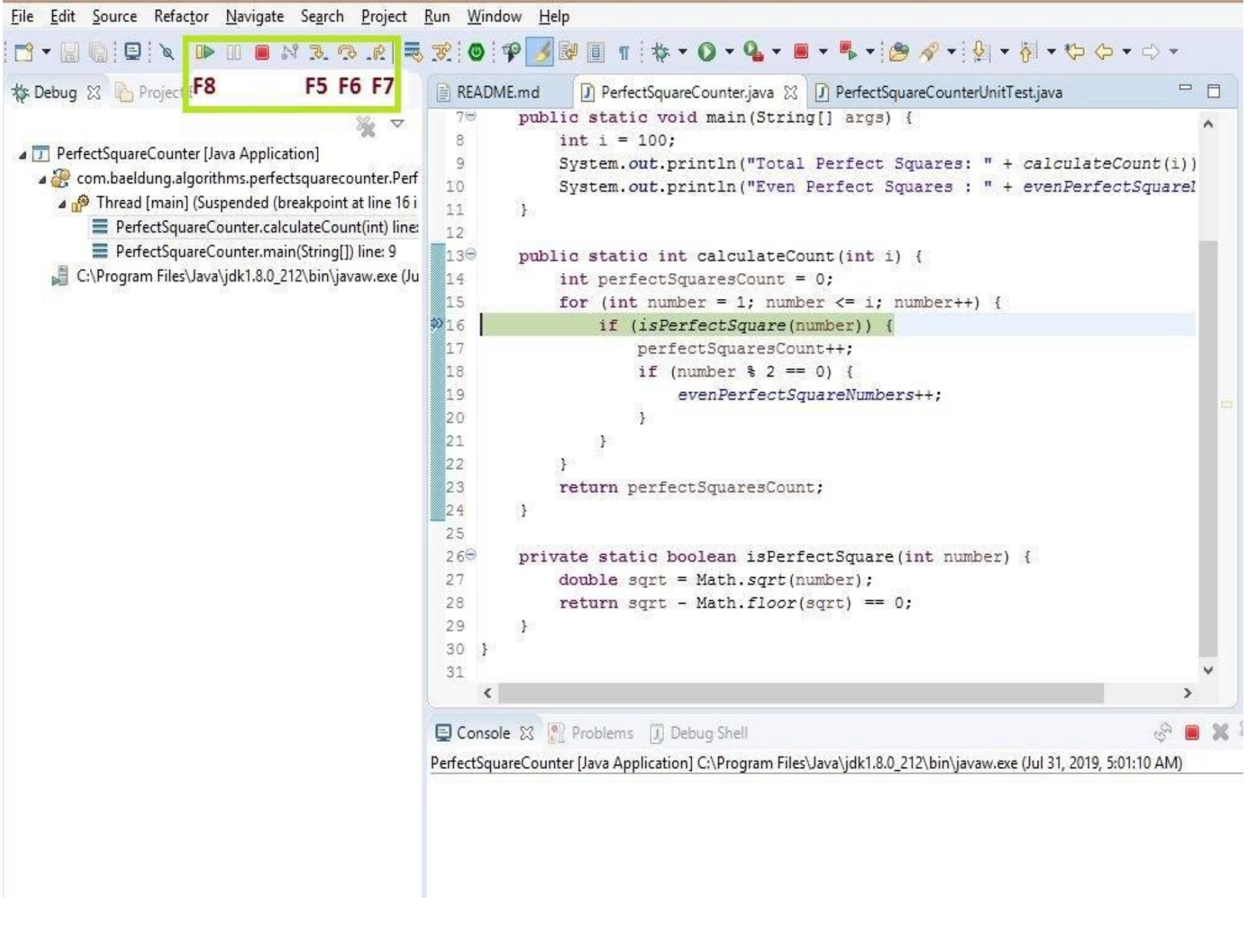
```
1  package com.baeldung.algorithms.perfectsquarecounter;
2
3  public class PerfectSquareCounter {
4
5      static int evenPerfectSquareNumbers = 0;
6
7      public static void main(String[] args) {
8          int i = 100;
9          System.out.println("Total Perfect Squares: " + calculateCount(i));
10         System.out.println("Even Perfect Squares : " + evenPerfectSquareNumbers);
11     }
12
13     public static int calculateCount(int i) {
14         int perfectSquaresCount = 0;
15         for (int number = 1; number <= i; number++) {
                   r)) {
                   ;
                   {
               Numbers++;
```

- Toggle Breakpoint Ctrl+Shift+B
- Disable Breakpoint Shift+Double Click
- Run to Line Ctrl+Alt+Click
- Go to Annotation Ctrl+1
- Validate
- Add Bookmark...
- Add Task...
- Show Quick Diff Ctrl+Shift+Q
- Show Revision Information
- Show Line Numbers
- Folding
- Preferences...
- Breakpoint Properties... Ctrl+Double Click

rce	Path	Location	Type

Code Control Flow

- Now that the debugger stops at the given breakpoints, we can proceed with further execution.
- Let's assume that the debugger is currently positioned as per the below screenshot, at Line 16:


File Edit Source Refactor Navigate Search Project Run Window Help
F8
F5 F6 F7
Debug
PerfectSquareCounter [Java Application]
com.baeldung.algorithms.perfectsquarecounter.Perf
Thread [main] (Suspended (breakpoint at line 16 i
PerfectSquareCounter.calculateCount(int) line:
PerfectSquareCounter.main(String[]) line: 9
C:\Program Files\Java\jdk1.8.0_212\bin\javaw.exe (Ju
README.md
PerfectSquareCounter.java
PerfectSquareCounterUnitTest.java
7 public static void main(String[] args) {
8 int i = 100;
9 System.out.println("Total Perfect Squares: " + calculateCount(i))
10 System.out.println("Even Perfect Squares : " + evenPerfectSquareI
11 }
12
13 public static int calculateCount(int i) {
14 int perfectSquaresCount = 0;
15 for (int number = 1; number <= i; number++) {
16 if (isPerfectSquare(number)) {
17 perfectSquaresCount++;
18 if (number % 2 == 0) {
19 evenPerfectSquareNumbers++;
20 }
21 }
22 }
23 return perfectSquaresCount;
24 }
25
26 private static boolean isPerfectSquare(int number) {
27 double sqrt = Math.sqrt(number);
28 return sqrt - Math.floor(sqrt) == 0;
29 }
30 }
31
Console
Problems
Debug Shell
PerfectSquareCounter [Java Application] C:\Program Files\Java\jdk1.8.0_212\bin\javaw.exe (Jul 31, 2019, 5:01:10 AM)

Debug Options

- The most commonly used debug options are:
- ***Step Into (F5)*** **– This operation goes inside the methods** used in the current line (if any); else, it proceeds to the next line. In this example, it will take the debugger inside the method *isPerfectSquare()*
- ***Step Over (F6)*** **– This operation processes the current line and proceeds to the next line.** In this example, this will execute the method *isPerfectSquare()* and proceed to the next line
- ***Step Return (F7)*** **– This operation finishes the current method and takes us back to the calling method.** Since in this case, we have a breakpoint in the loop, it will be still within the method, else it would go back to the main method
- ***Resume (F8)*** **– This operation will simply continue with the execution until the program ends** unless we hit any further breakpoint

Variables

- We can see the values of variables during the execution under the Variables view. In order to see the static variables, we can select the drop-down option *Java -> Show Static Variables*.
- **Using the variables view, it's possible to change any value to the desired value during the execution.**
- For example, if we need to skip a few numbers and directly start with the number 80, we could do that by changing the value of the variable *number*

README.md | PerfectSquareCounter.java | PerfectSquareCounterUnitTest.java

```
7    public static void main(String[] args) {
8        int i = 100;
9        System.out.println("Total Perfect Squares: " + calculateCount(i));
10       System.out.println("Even Perfect Squares : " + evenPerfectSquareN
11   }
12
13   public static int calculateCount(int i) {
14       int perfectSquaresCount = 0;
15       for (int number = 1; number <= i; number++) {
16           if (isPerfectSquare(number)) {
17               perfectSquaresCount++;
18               if (number % 2 == 0) {
19                   evenPerfectSquareNumbers++;
20               }
21           }
22       }
23       return perfectSquaresCount;
24   }
25
26   private static boolean isPerfectSquare(int number) {
27       double sqrt = Math.sqrt(number);
28       return sqrt - Math.floor(sqrt) == 0;
29   }
30 }
```

(x)= Variables | Breakpoints | Expressions

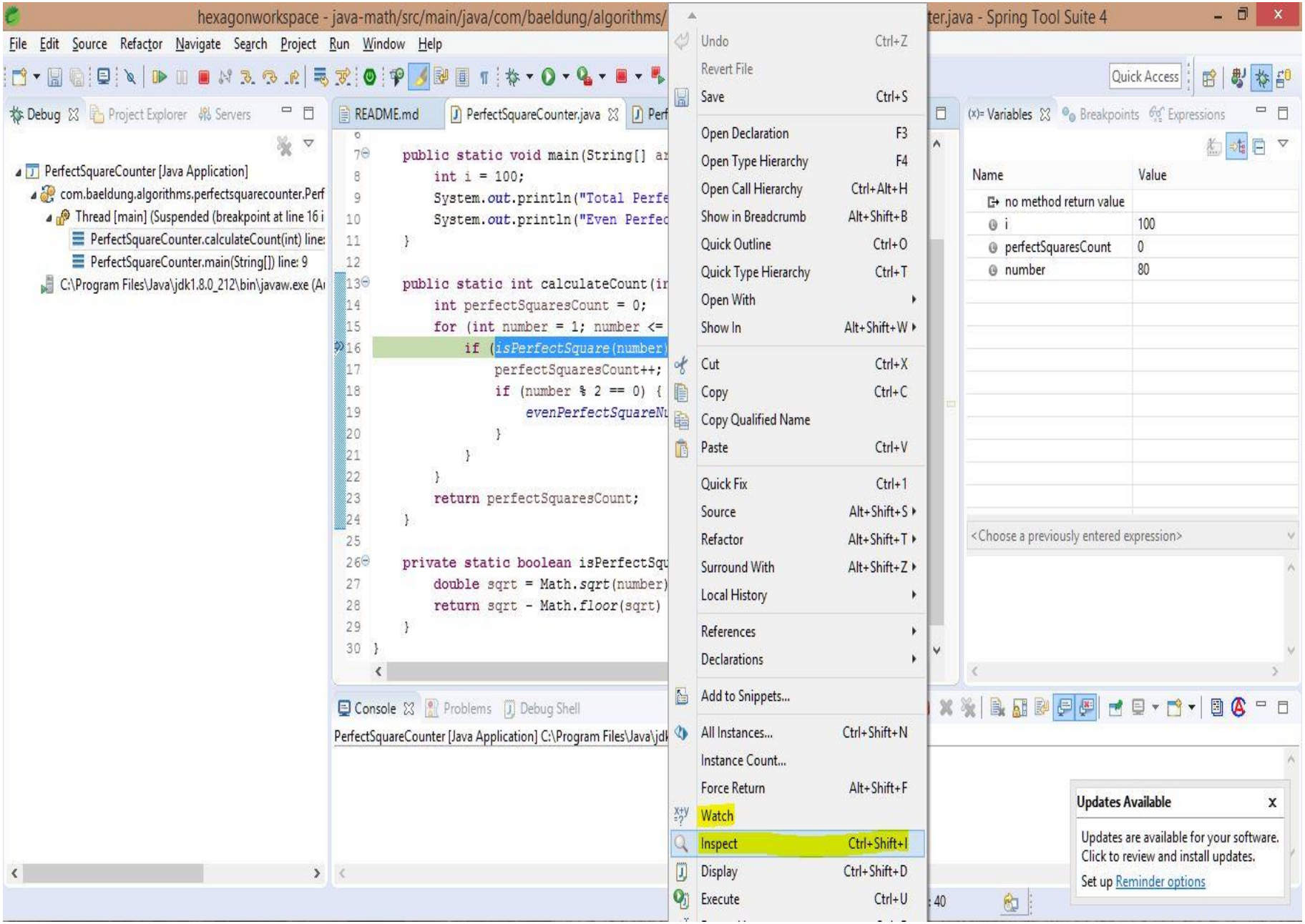
Name	Value
no method return value	
i	100
perfectSquaresCount	0
number	80

<Choose a previously entered expression>

80

Inspecting value

- If we need to inspect the value of a Java expression or statement, we can select the particular expression in the editor, right-click, and Inspect, as shown below. A handy shortcut is to **hit *Ctrl+Shift+I* on the expression to see the value:**


hexagonworkspace - java-math/src/main/java/com/baeldung/algorithms/
ter.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Debug
Project Explorer
Servers
PerfectSquareCounter [Java Application]
com.baeldung.algorithms.perfectsquarecounter.Perf
Thread [main] (Suspended (breakpoint at line 16 i
PerfectSquareCounter.calculateCount(int) line:
PerfectSquareCounter.main(String[]) line: 9
C:\Program Files\Java\jdk1.8.0_212\bin\javaw.exe (A
README.md
PerfectSquareCounter.java
Perf
public static void main(String[] ar
int i = 100;
System.out.println("Total Perfe
System.out.println("Even Perfec
}
public static int calculateCount(ir
int perfectSquaresCount = 0;
for (int number = 1; number <=
if (isPerfectSquare(number)
perfectSquaresCount++;
if (number % 2 == 0) {
evenPerfectSquareNu
}
}
}
return perfectSquaresCount;
}
private static boolean isPerfectSqu
double sqrt = Math.sqrt(number)
return sqrt - Math.floor(sqrt)
}
}
Console
Problems
Debug Shell
PerfectSquareCounter [Java Application] C:\Program Files\Java\jdk
(x)= Variables
Breakpoints
Expressions
Name
Value
no method return value
i
100
perfectSquaresCount
0
number
80
<Choose a previously entered expression>
Undo Ctrl+Z
Revert File
Save Ctrl+S
Open Declaration F3
Open Type Hierarchy F4
Open Call Hierarchy Ctrl+Alt+H
Show in Breadcrumb Alt+Shift+B
Quick Outline Ctrl+O
Quick Type Hierarchy Ctrl+T
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Quick Fix Ctrl+1
Source Alt+Shift+S
Refactor Alt+Shift+T
Surround With Alt+Shift+Z
Local History
References
Declarations
Add to Snippets...
All Instances... Ctrl+Shift+N
Instance Count...
Force Return Alt+Shift+F
Watch
Inspect Ctrl+Shift+I
Display Ctrl+Shift+D
Execute Ctrl+U
Updates Available
Updates are available for your software.
Click to review and install updates.
Set up Reminder options

Lombok

- The project Lombok is a popular and widely used Java library that is used to minimize or remove the boilerplate code. It saves time and effort. Just by using the annotations, we can save space and readability of the source code. It is automatically plugging into IDEs and build tools to spice up our Java application.

Features of Lombok

- It reduces the **boilerplate**
- It replaces boilerplate code with easy-to-use **annotations**.
- It makes code easier to read and **less error-prone**.
- By using Lombok the developers becomes more **productive**.
- It works well with all popular IDEs.
- Also provides **delombok** utility (adding back all the boilerplate code).
- Provide annotation for checking null values.
- Concise data objects
- Easy cleanup
- Locking safely
- Effortless logging

Annotations in Lombok

1. @NoArgsConstructor
2. @AllArgsConstructor
3. @ToString
4. @Getter @Setter

Exception Handling

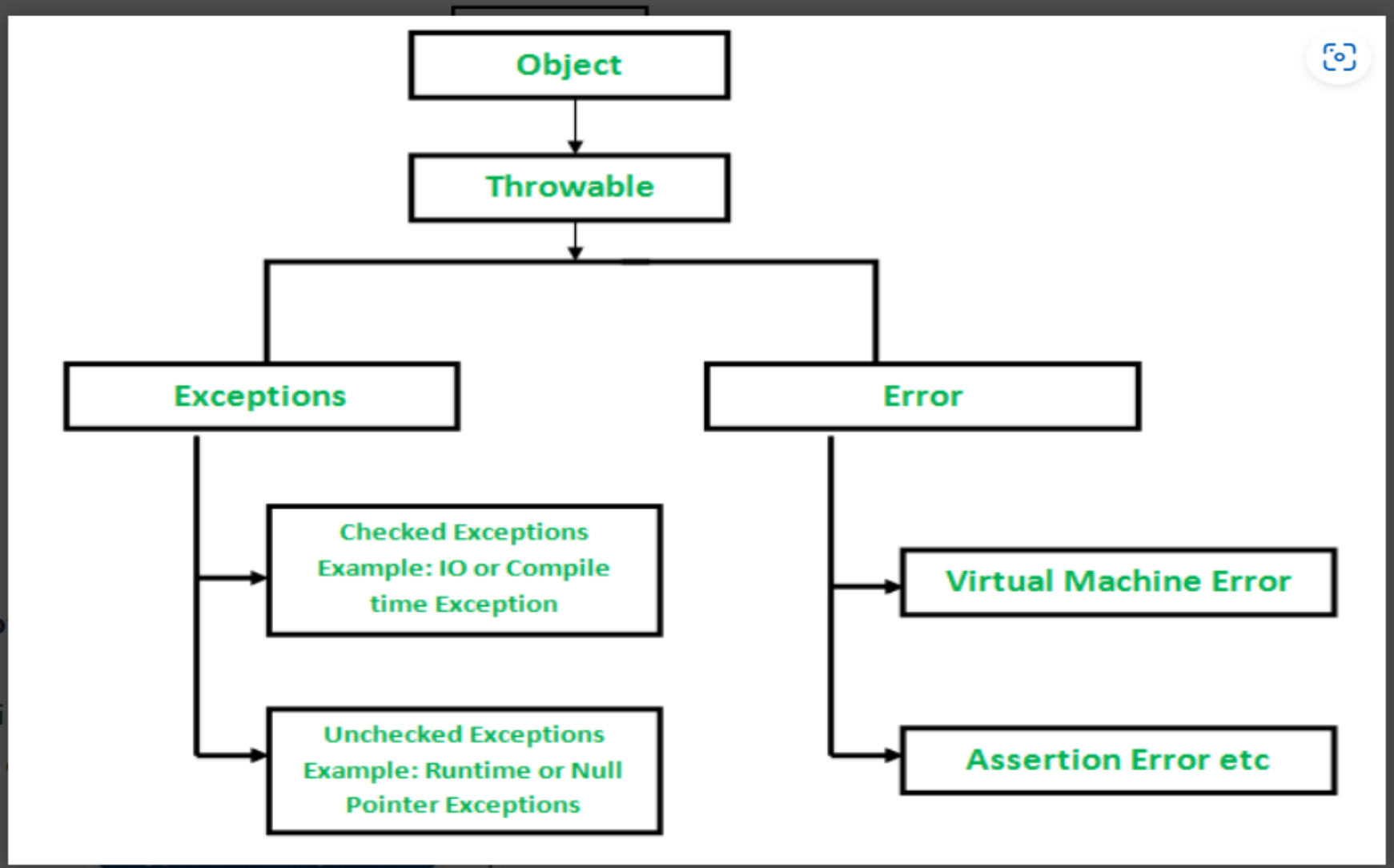
- **Exception Handling** in Java is one of the effective means to handle the runtime errors so that the regular flow of the application can be preserved. Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc

- **Exception** is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exceptions can be caught and handled by the program. When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception occurred.

- **Major reasons why an exception Occurs**
- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

- **Errors** represent irrecoverable conditions such as Java virtual machine (JVM) running out of memory, memory leaks, stack overflow errors, library incompatibility, infinite recursion, etc. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

- All exception and error types are subclasses of class **Throwable**, which is the base class of the hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.


Object
Throwable
Exceptions
Error
Checked Exceptions
Example: IO or Compile
time Exception
Unchecked Exceptions
Example: Runtime or Null
Pointer Exceptions
Virtual Machine Error
Assertion Error etc

- Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.

- **Exceptions can be categorized in two ways:**

1.Built-in Exceptions

1. Checked Exception
2. Unchecked Exception

- **Built-in Exceptions:**
- Built-in exceptions are the exceptions that are available in Java libraries. These exceptions are suitable to explain certain error situations.
- **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

- **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

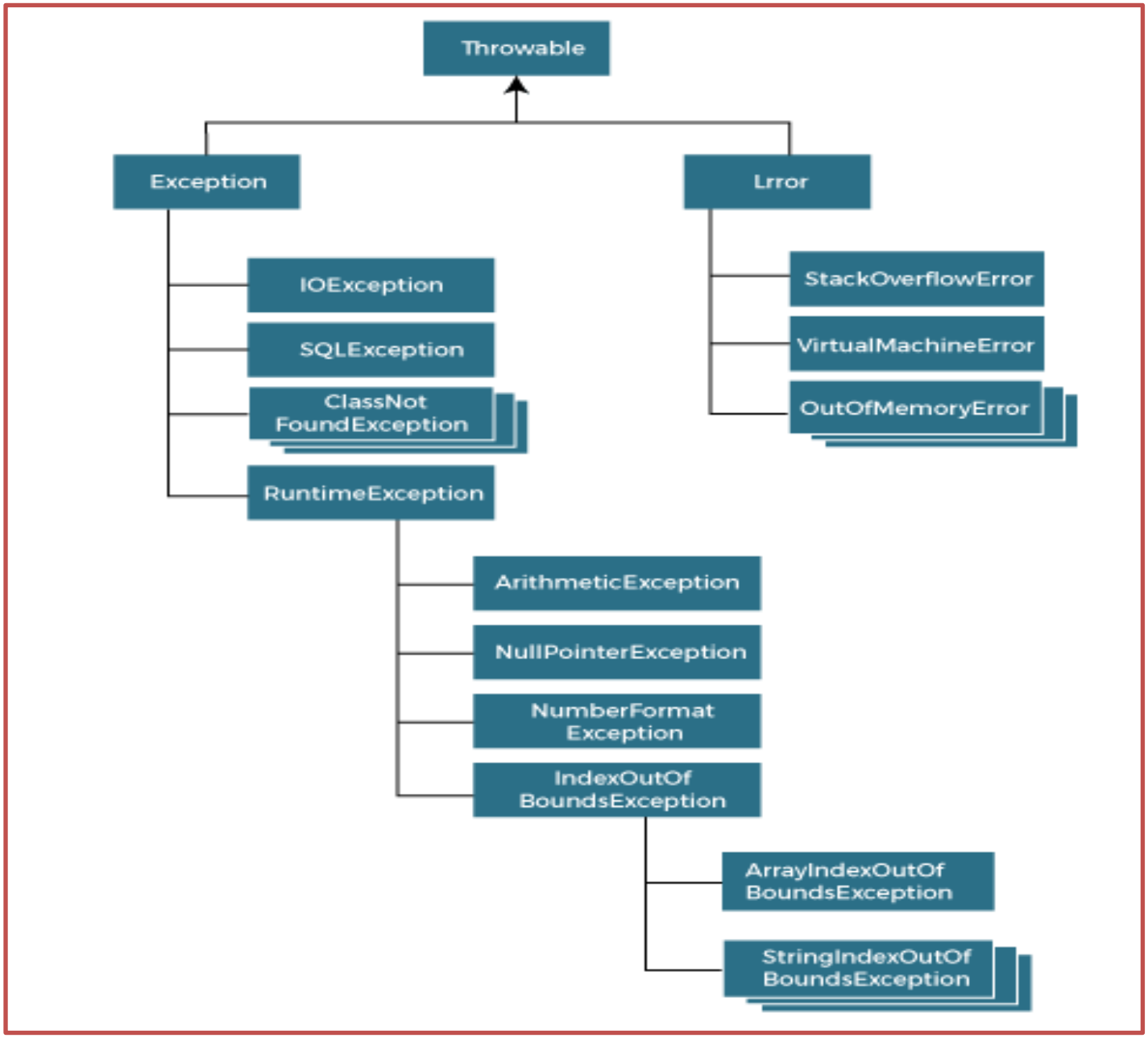
- **User-Defined Exceptions:**
- Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called ‘user-defined Exceptions’.
- The ***advantages of Exception Handling in Java*** are as follows:

1. Provision to Complete Program Execution
2. Easy Identification of Program Code and Error-Handling Code
3. Propagation of Errors
4. Meaningful Error Reporting
5. Identifying Error Types

- **Methods to print the Exception information:**
- **1.printStackTrace()**– This method prints exception information in the format of Name of the exception: description of the exception, stack

Handling Exception

- Try block:This is the block that contains code that can throw up exception
- Catch block:This is the block that contains exception handling code
- Finally block:This block contains code that will get executed in all scenarios
- Throw : This key word is used to throw Exception and it’s sub class objects
- Throws : This keyword is used for declaring that a function or constructor throws exception


Throwable
Exception
Lrror
IOException
SQLException
ClassNot FoundException
RuntimeException
StackOverflowError
VirtualMachineError
OutOfMemoryError
ArithmeticException
NullPointerException
NumberFormat Exception
IndexOutOf BoundsException
ArrayIndexOutOf BoundsException
StringIndexOutOf BoundsException

Abstract Class

- A class declared as abstract may or may not contain abstract methods
- An abstract method does not have any body
- The purpose of abstract class is for inheritance
- The class inheriting abstract class has to define all the abstract methods ,or it has to be declared as abstract

Interface

- An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class. An interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.
- The interface in Java is *a* mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**.

Interface

- When we decide a type of entity by its behaviour and not via attribute we should define it as an interface.
- Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).

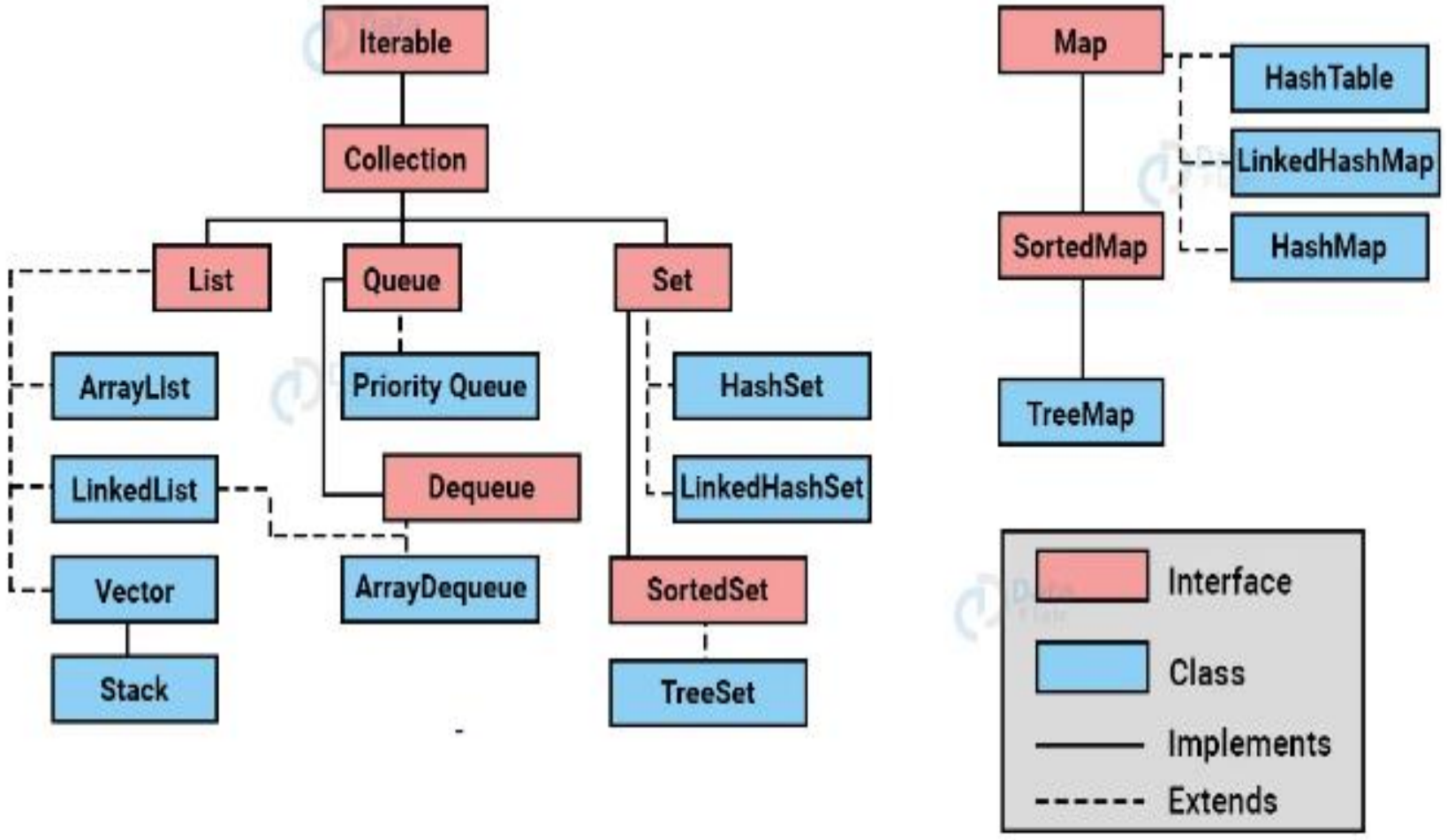
Collection

- The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Collection

- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Hierarchy of Collection Framework in Java



Set Interface

- Set is used to create collection of unordered unique values
- Set set=new HashSet();

```
set.add(10);
set.add(20);
set.add(11);
set.add(19);
```

List

- List is used to create collection of ordered values .List can contain duplicate values

- List list =new ArrayList();

```
list.add(12);
list.add(23);
list.add(19);
list.add(22);
list.add(19);
```

Map

- A map contains group of Key Value pairs
- Each Key and value pair is known as an entry
- All the keys in the map have to be unique