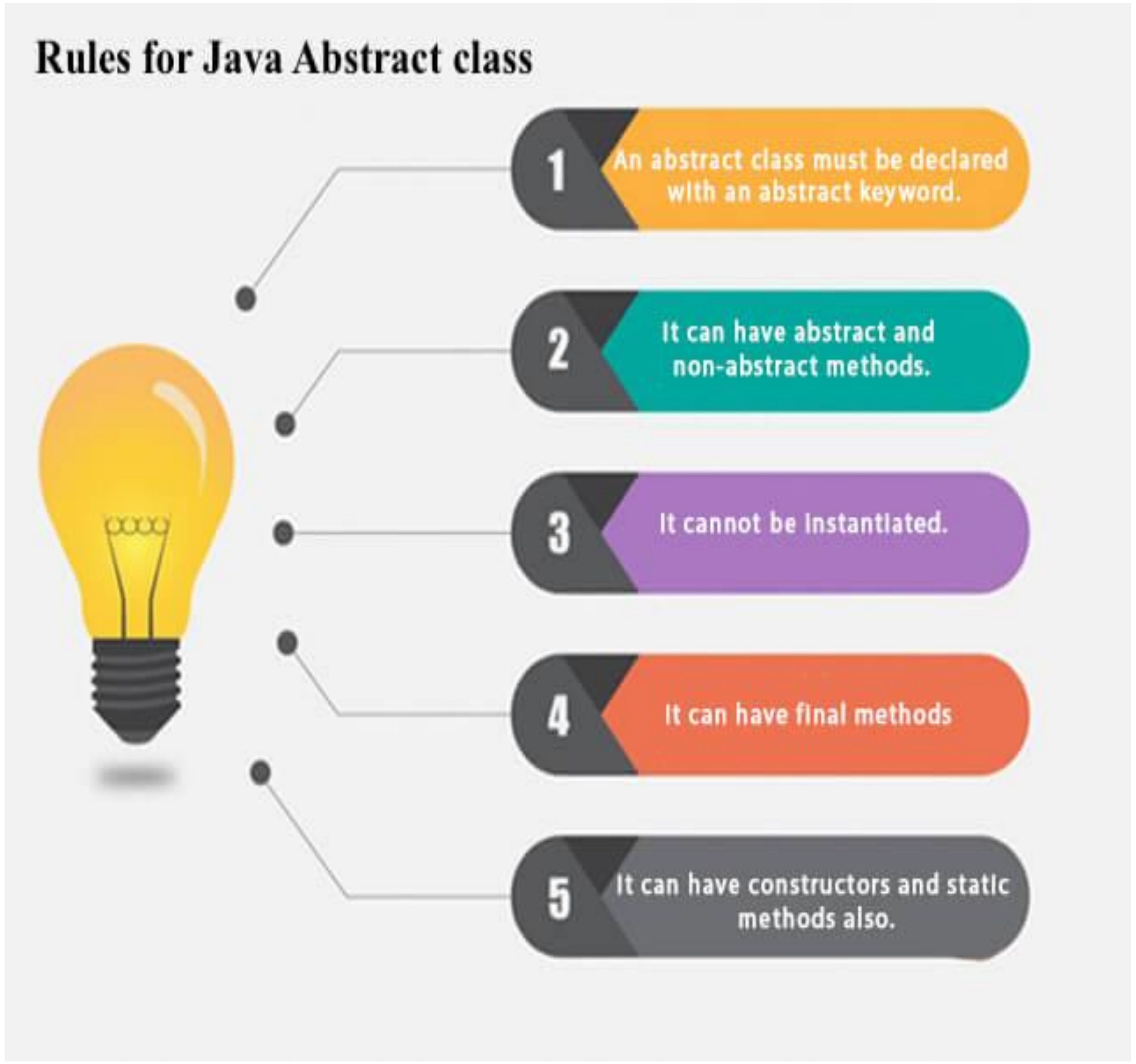
Welcome to Java

Abstraction

- **Deferred Implementation** (Abstract Class and Interface)
- **Abstraction** is the quality of dealing with ideas rather than events.
- Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user.
- Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.
- In java, abstraction is achieved by interfaces and abstract classes. We can achieve 100% abstraction using interfaces.


Rules for Java Abstract class

1. An abstract class must be declared with an abstract keyword.
2. It can have abstract and non-abstract methods.
3. It cannot be instantiated.
4. It can have final methods
5. It can have constructors and static methods also.

Abstraction

Abstract classes and Abstract methods :

- An abstract class is a class that is declared with abstract keyword.
- An abstract method is a method that is declared without an implementation.
- An abstract class may or may not have all abstract methods. Some of them can be concrete methods
- A method defined abstract must always be redefined in the subclass,thus making overriding compulsory OR either make subclass itself abstract.
- Any class that contains one or more abstract methods must also be declared with abstract keyword.
- There can be no object of an abstract class.That is, an abstract class can not be directly instantiated with the *new operator*.
- An abstract class can have parametrized constructors and default constructor is always present in an abstract class.

Interfaces

- Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).
- Interfaces specify what a class must do and not how.
- Why Do we use Interface
    - It is used to achieve total abstraction.
    - Since java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance .
    - It is also used to achieve loose coupling.
- Important points about interface:
    - We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
    - A class can implement more than one interface.
    - An interface can extends another interface or interfaces (more than one interface) .
    - A class that implements interface must implements all the methods in interface.
    - All the methods are public and abstract. And all the fields are public, static, and final.

Interfaces

- An **interface in Java** is a blueprint of a class. It has 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 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.
- It cannot be instantiated just like the abstract class.
- Since Java 8, we can have default and static methods in an interface.
- Since Java 9, we can have private methods in an interface.

Why use Java interface?

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Interfaces

- An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- A class that implements an interface must implement all the methods declared in the interface.

Syntax:

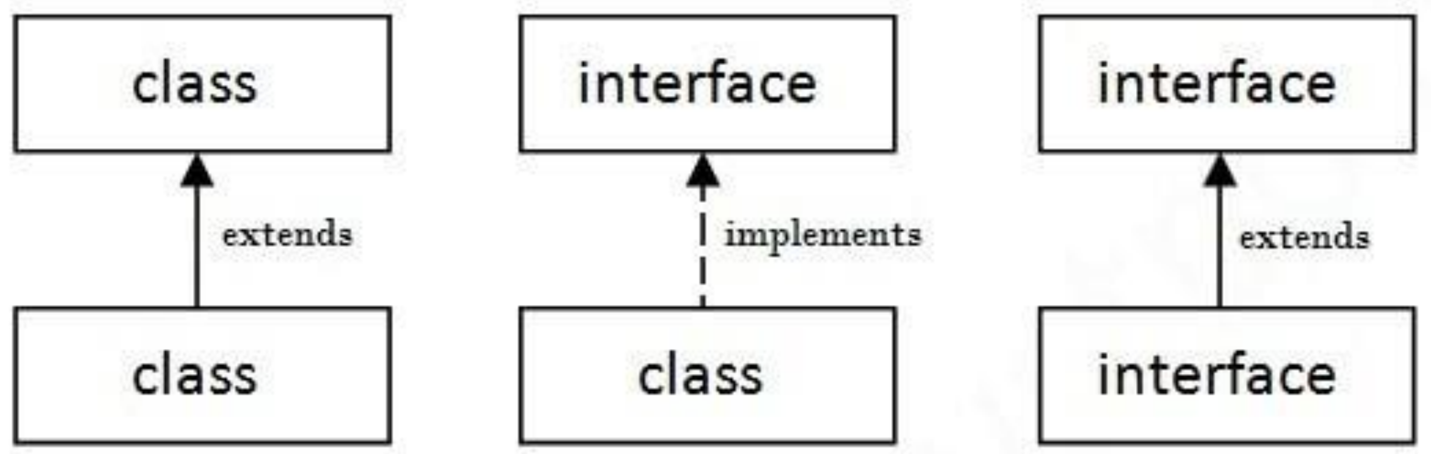
```
interface <interface_name>{

    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

Interfaces

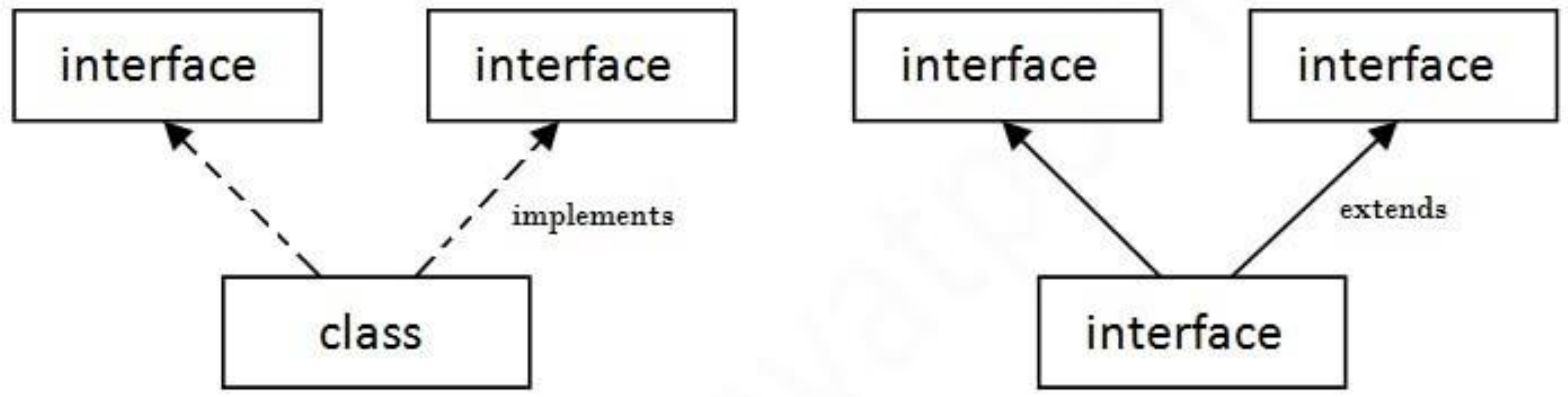
- The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.
- In other words, Interface fields are public, static and final by default, and the methods are public and abstract.

The relationship between Classes and Interfaces



Multiple Inheritance in Java by Interface

- If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

- **Interface Inheritance:** A class implements an interface, but one interface extends another interface.

Java 8 method in Interface

- Since Java 8, we can have method body in interface. But we need to make it default method.
- Since Java 8, we can have static method in interface.

- An interface which has no member is known as a marker or tagged interface, for example, Serializable, Cloneable, Remote, etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.
- An interface can have another interface which is known as a nested interface.

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```

Difference between abstract class and Interface

Abstract Class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Strings

- In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

  ```
  char[] ch={'j','a','v','a','t',’r',’a','i','n',’g'};

  String s=new String(ch);
  is same as:
  String s=“javatraing";
  ```

- **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.
- The java.lang.String class implements Serializable, Comparable and CharSequence interfaces.
- The CharSequence interface is used to represent the sequence of characters. String, StringBuffer and StringBuilder classes implement it. It means, we can create strings in java by using these three classes.

Strings

- The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.
- Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

- There are two ways to create String object:
    - By string literal
    - By new keyword

* Java String literal is created by using double quotes. For Example:

```
String s="welcome";
```

Strings

- Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.
- String objects are stored in a special memory area known as the "string constant pool".
- Java uses the concept of String Constant pool to make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

By New Keyword

String s=new String("Welcome");//creates two objects and one reference variable

- In such case, JVM will create a new string object in normal (non-pool) heap memory. The variable s will refer to the object in a heap (non-pool).

Immutable Strings

- In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.
- Once string object is created its data or state can't be changed but a new string object is created.

Why String Objects are immutable in Java

- Because java uses the concept of string literal. Suppose there are 5 reference variables, all refers to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

Compare Strings

- We can compare string in java on the basis of content and reference.
- It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.
- There are three ways to compare string in java:
  - By equals() method
  - By = = operator
  - By compareTo() method

String compare by equals() method

- The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:
  - **public boolean equals(Object another)** compares this string to the specified object.
  - **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

Compare Strings

String compare by == operator

- The = = operator compares references not values.

String compare by compareTo() method

- The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.
- Suppose s1 and s2 are two string variables. If:
  - **s1 == s2** :0
  - **s1 > s2** :positive value
  - **s1 < s2** :negative value

String Concatenation

- In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:
  - By + (string concatenation) operator
  - By concat() method

- In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also.

Substring

- A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.
- You can get substring from the given string object by one of the two methods:
- **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
- **public String substring(int startIndex, int endIndex):** This method returns new String object containing the substring of the given string from specified startIndex to endIndex.
  - startIndex is inclusive
  - endIndex is exclusive

String Class Methods

- The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.
- Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.
    - toUpperCase()
    - toLowerCase()
    - trim()
    - startsWith()
    - endsWith()
    - charAt()
    - length()

String Class Methods

- **toCharArray() -** The java string toCharArray() method converts this string into character array. It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.
- **split() -** The java string split() method splits this string against given regular expression and returns a char array.
- **replaceAll() -** The java string replaceAll() method returns a string replacing all the sequence of characters matching regex and replacement string.
- **lastIndexOf() -** The java string lastIndexOf() method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

String Class Methods

- **join()** - The java string join() method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.
  - In case of null element, "null" is added. The join() method is included in java string since JDK 1.8.
  - There are two types of join() methods in java string.

- **format() -** The java string format() method returns the formatted string by given locale, format and arguments.
  - If you don't specify the locale in String.format() method, it uses default locale by calling Locale.getDefault() method.
  - The format() method of java language is like sprintf() function in c language and printf() method of java language.

StringBuffer

- Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
- Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

Important Constructors in StringBuffer:

Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

Importanat Methods in StringBuffer

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.

Importanat Methods in StringBuffer

Modifier and Type	Method	Description
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	is used to return the character at the specified position.

Importanat Methods in StringBuffer

Modifier and Type	Method	Description
public int	length()	is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String	substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

Mutable String

- A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

- **StringBuffer Capacity()** - the capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.
- **ensureCapacity() -** The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

StringBuilder

- Java StringBuilder class is used to create mutable (modifiable) string. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5

Important Constructors of StringBuilder Class

Constructor	Description
StringBuilder()	creates an empty string Builder with the initial capacity of 16.
StringBuilder(String str)	creates a string Builder with the specified string.
StringBuilder(int length)	creates an empty string Builder with the specified capacity as length.

Importanat Methods in StringBuilder

Method	Description
public StringBuilder append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public StringBuilder insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public StringBuilder replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public StringBuilder delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.

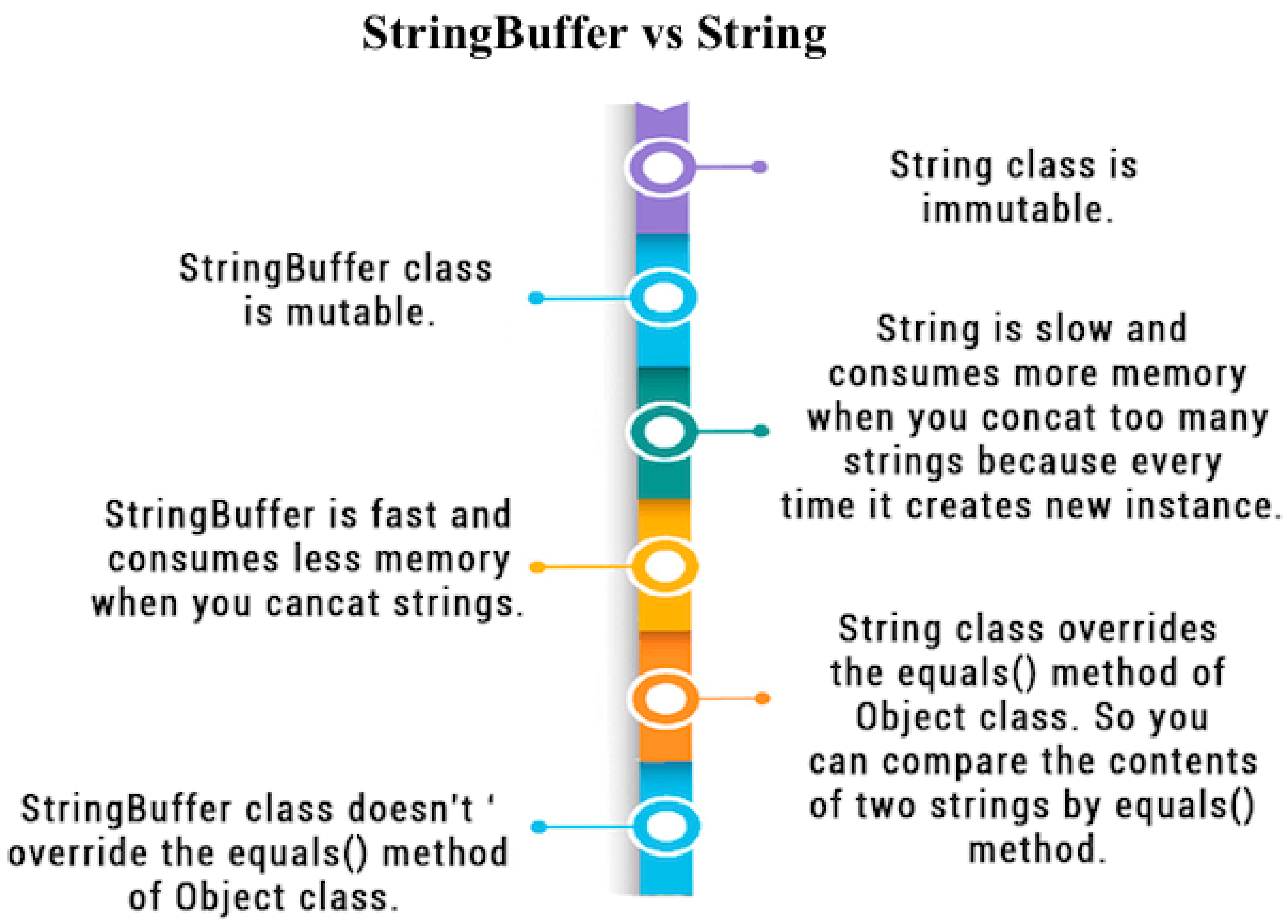
Importanat Methods in StringBuilder

Method	Description
public StringBuilder reverse()	is used to reverse the string.
public int capacity()	is used to return the current capacity.
public void ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char charAt(int index)	is used to return the character at the specified position.
public int length()	is used to return the length of the string i.e. total number of characters.

Importanat Methods in StringBuilder

Method	Description
public String substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String substring(int beginIndex, int endIndex)	is used to return the substring from the specified beginIndex and endIndex.

StringBuffer vs String



StringBuffer vs String

String class is immutable.

StringBuffer class is mutable.

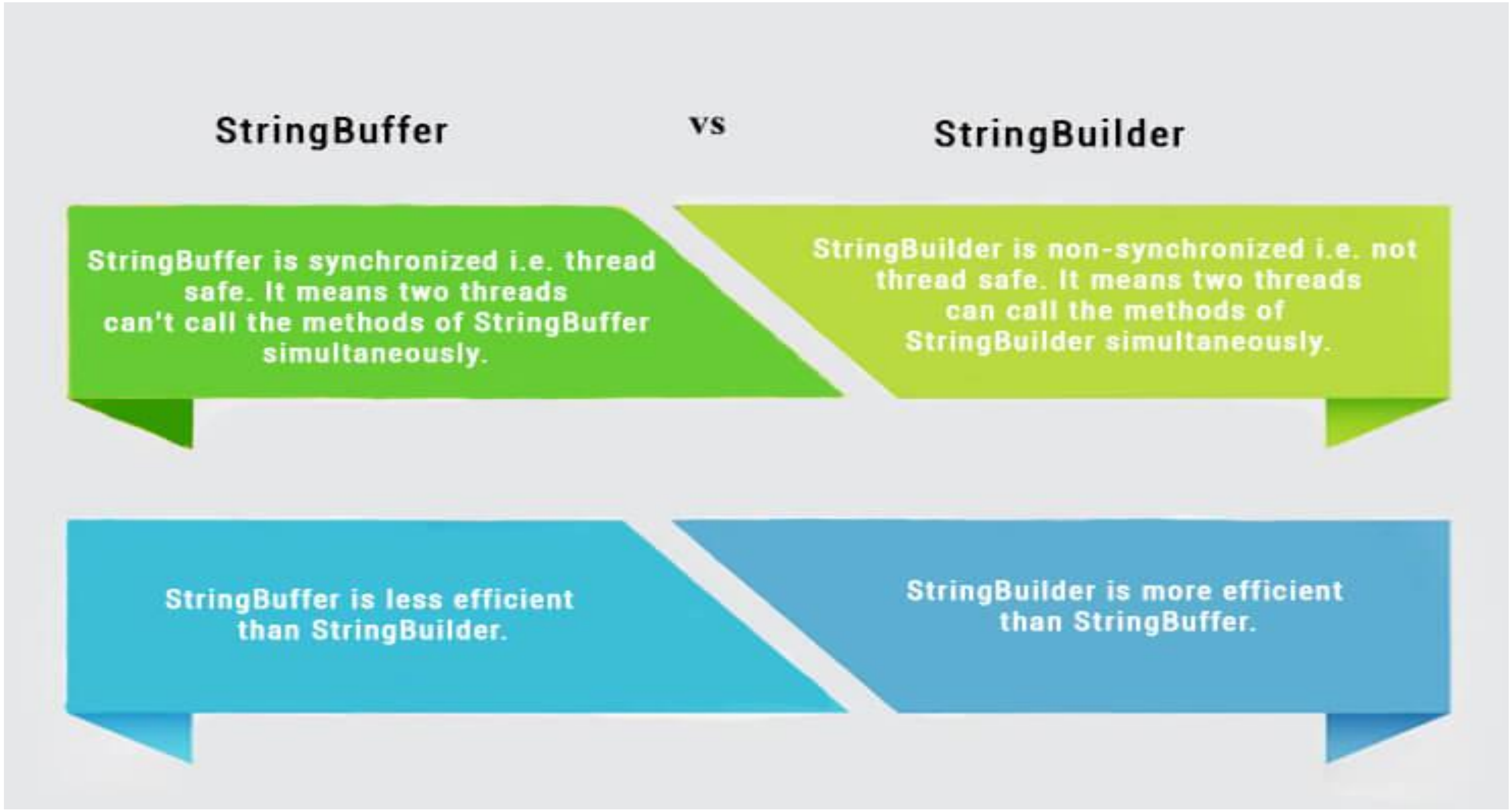
String is slow and consumes more memory when you concat too many strings because every time it creates new instance.

StringBuffer is fast and consumes less memory when you cancat strings.

String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.

StringBuffer class doesn't ' override the equals() method of Object class.

StringBuffer vs StringBuilder



StringBuffer vs StringBuilder

StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.

StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.

StringBuffer is less efficient than StringBuilder.

StringBuilder is more efficient than StringBuffer.

StringTokenizer Classes

- StringTokenizer class in Java is used to break a string into tokens.
- A StringTokenizer object internally maintains a current position within the string to be tokenized.
- Some operations advance this current position past the characters processed. A token is returned by taking a substring of the string that was used to create the StringTokenizer object. It provides the first step in the parsing process often called lexer or scanner.
- The String Tokenizer class allows an application to break strings into tokens. It implements the Enumeration interface. This class is used for parsing data.
- To use String Tokenizer class we have to specify an input string and a string that contains delimiters. Delimiters are the characters that separate tokens. Each character in the delimiter string is considered a valid delimiter. Default delimiters are whitespaces, new line, space, and tab.

StringTokenizer Classes

- Constructors of StringToken: Let us consider ‘str’ as the string to be tokenized:
  - **StringTokenizer(String str):** default delimiters like newline, space, tab, carriage return, and form feed.
  - **StringTokenizer(String str, String delim)**: delim is a set of delimiters that are used to tokenize the given string.
  - **StringTokenizer(String str, String delim, boolean flag)**: The first two parameters have the same meaning wherein The flag serves the following purpose.

Difference between StringTokenizer and Split in Java

StringTokenizer	Split
It is a legacy class that allows an application to break a string into tokens.	It is a method of the String class or the java.util.regex package that splits this string around matches of the given regular expression.
It returns one substring at a time.	It returns an array of substrings.
It can’t handle empty strings well.	It can handle empty strings when you need to parse empty tokens like ant, bat, pat
It is comparatively less robust & syntactically fussy.	It is more robust & has an easy syntax.
It just accepts a String by which it will split the string	It accepts regular expressions.
The delimiter is just a character long.	The delimiter is a regular expression.
It is essentially designed for pulling out tokens delimited by fixed substrings.	It is essentially designed to parse text data from a source outside your program, like from a file, or from the user.
Because of this restriction, it’s about twice as fast as split().	Slower than StringTokeniser
Consists of a constructor with a parameter that allows you to specify possible delimiter	No constructor.

Exception Handling

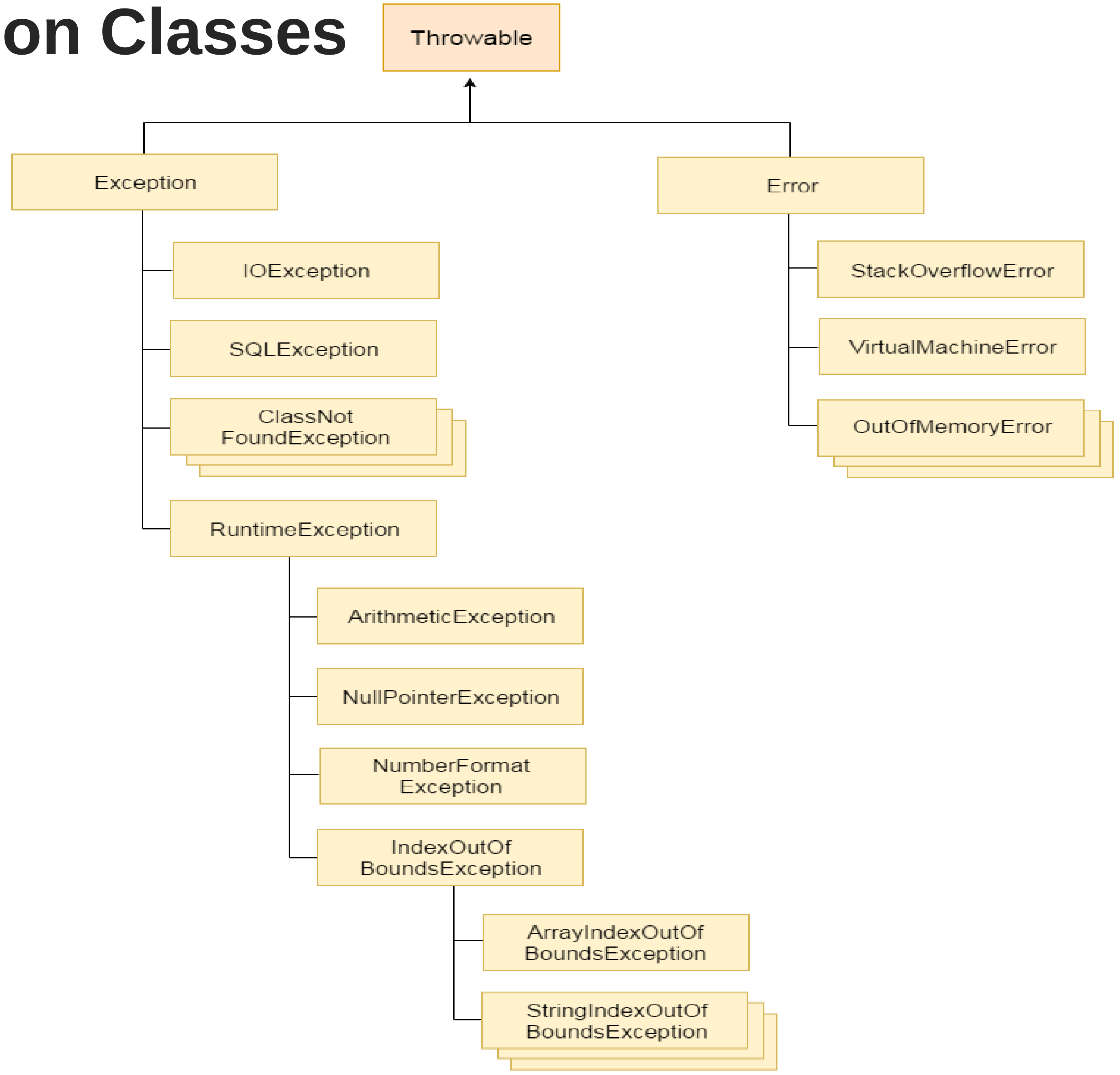
- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.
- Exception is an abnormal condition.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage of Exception Handling

- The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling.

Hierarchy of Java Exception Classes

- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses:
  - Exception
  - Error



Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:
  - Checked Exception
  - Unchecked Exception
  - Error

Unchecked Exception
Checked Exception
Error
Exception Handling

Difference between Checked and Unchecked Exceptions

Checked Exception

- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

Unchecked Exception

- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

Error

- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Common Scenorios of Java Exceptions

A scenario where ArithmeticException occurs

- If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

A scenario where NullPointerException occurs

- If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

A scenario where NumberFormatException occurs

- The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

Common Scenorios of Java Exceptions

A scenario where ArrayIndexOutOfBoundsException occurs

- If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java Exceptions Index

- Java Try-Catch Block
- Java Multiple Catch Block
- Java Nested Try
- Java Finally Block
- Java Throw Keyword
- Java Exception Propagation
- Java Throws Keyword
- Java Throw vs Throws
- Java Exception Handling with Method Overriding
- Java Custom Exceptions

Try-catch block

try block

- Java **try** block is used to enclose the code that might throw an exception. It must be used within the method.
- If an exception occurs at the particular statement of try block, the rest of the block code will not execute. So, it is recommended not to keeping the code in try block that will not throw an exception.
- Java try block must be followed by either catch or finally block.

Syntax of Java try-catch

```
try{
//code that may throw an exception
}catch(Exception_class_Name ref){}
```

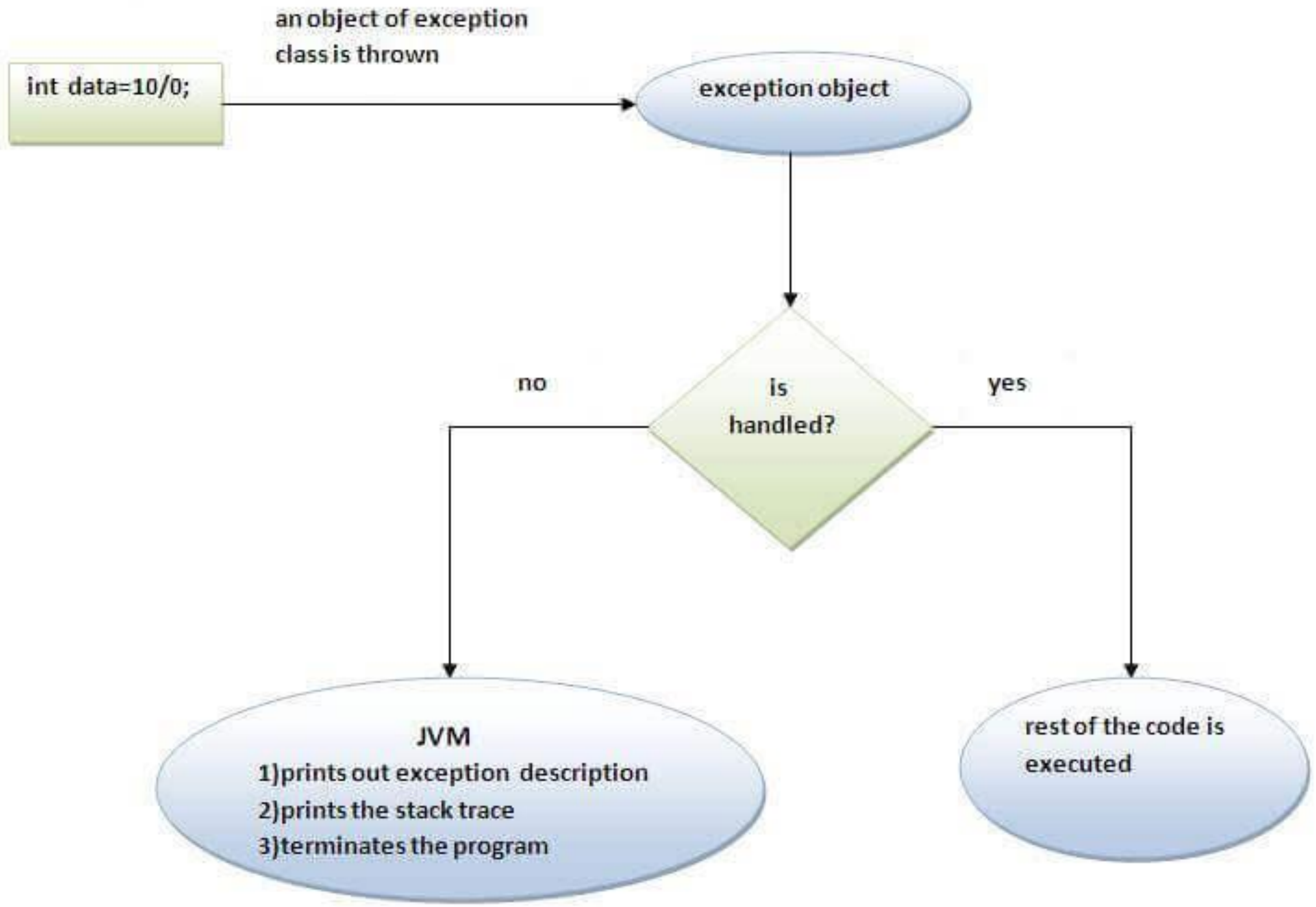
Syntax of try-finally block

```
try{
//code that may throw an exception
}finally{}
```

Catch block

- Java catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.
- The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Internal working of java try-catch block


an object of exception class is thrown
int data=10/0;
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Java Catch Multiple Exceptions

- A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

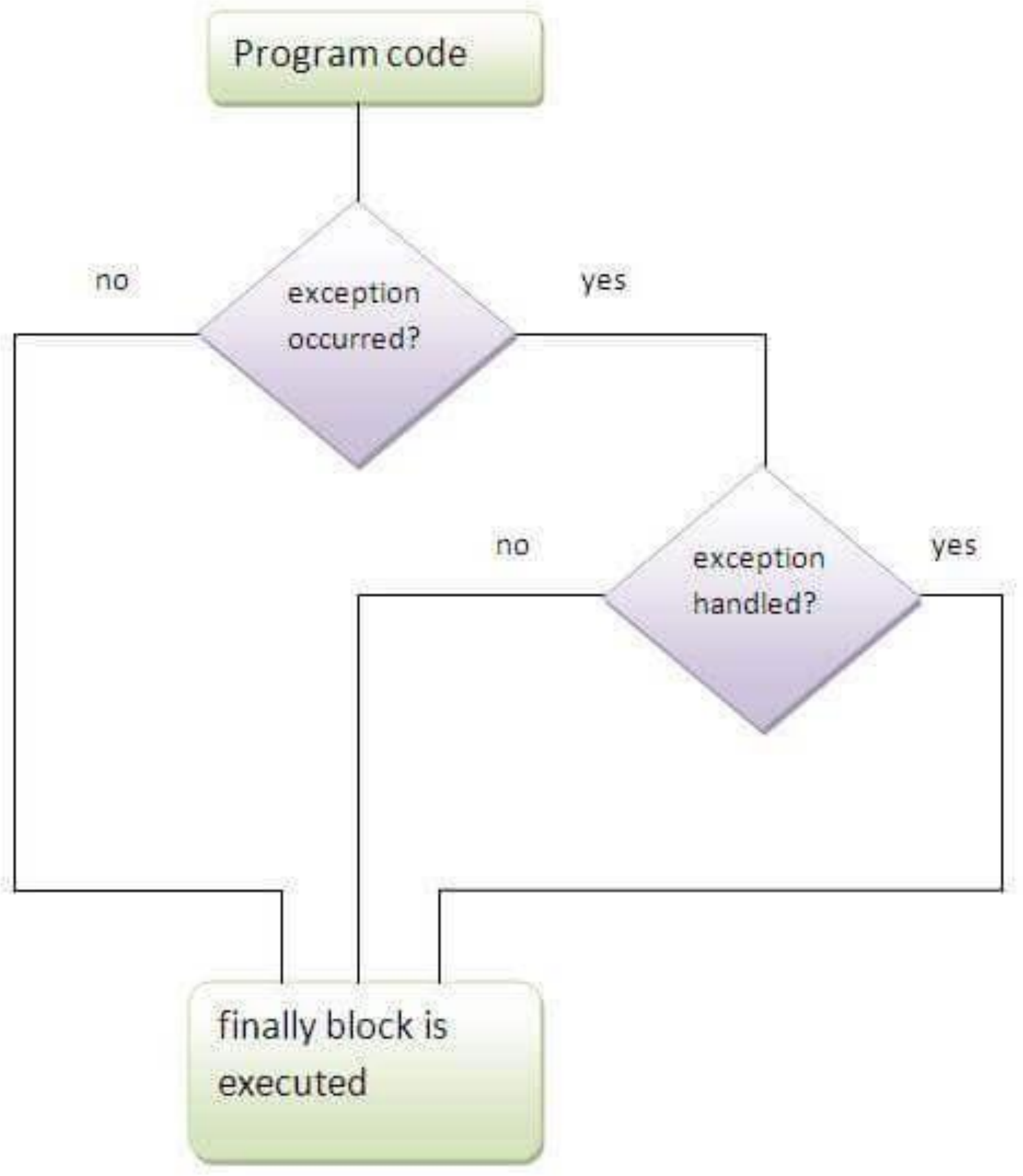
Nested try block

- The try block within a try block is known as nested try block in java.
- **Why use Nested try:** Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

```
try
{
    statement 1;
    try
    {
        statement 1;
    }
    catch(Exception e)
    { }
}
catch(Exception e)
{ }
```

finally block

- **finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.
- For each try block there can be zero or more catch blocks, but only one finally block.
- The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).



Throw Exception

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.
- Syntax is

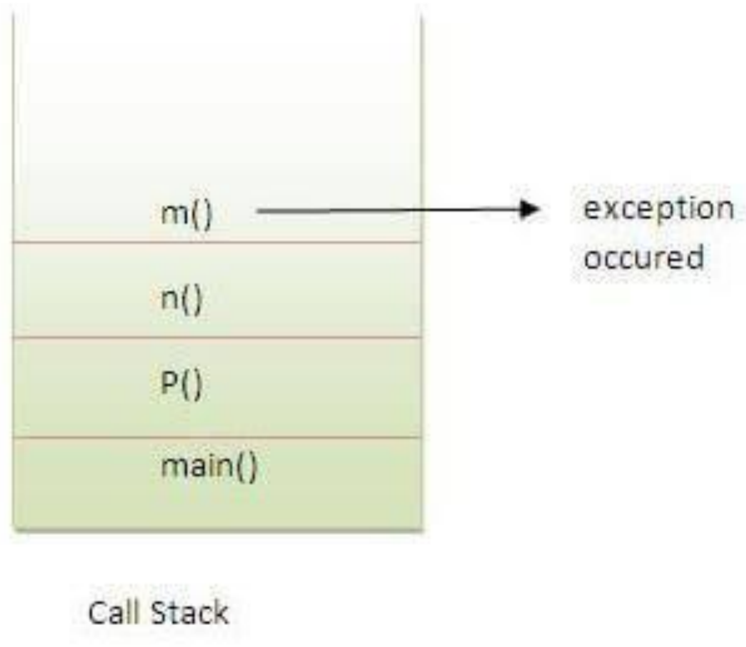
```
        throw exception;
```

throw IOException.

```
        throw new IOException("sorry device error);
```

Exception Propagation

- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.
- By default Unchecked Exceptions are forwarded in calling chain (propagated).
- By default, Checked Exceptions are not forwarded in calling chain (propagated).



throws keyword

- The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax

```
return_type method_name() throws exception_class_name{
        //method code
}
```

- **Advantage:**
  - Now Checked Exception can be propagated (forwarded in call stack).
  - It provides information to the caller of the method about the exception.

throws keyword

- If you are calling a method that declares an exception, There are two cases:
- **Case1:**You caught the exception i.e. handle the exception using try/catch.
- **Case2:**You declare the exception i.e. specifying throws with the method.

Difference between throw and throws in Java

No	Throw	throws
1	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3	Throw is followed by an instance.	Throws is followed by class.
4	Throw is used within the method.	Throws is used with the method signature.
5	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Exception Handling with Method Overriding

There are many rules if we talk about method overriding with exception handling.
The Rules are as follows:

- If the superclass method does not declare an exception
    - If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
- If the superclass method declares an exception
    - If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Custom Exceptions

- If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.
- By the help of custom exception, you can have your own exception and message.

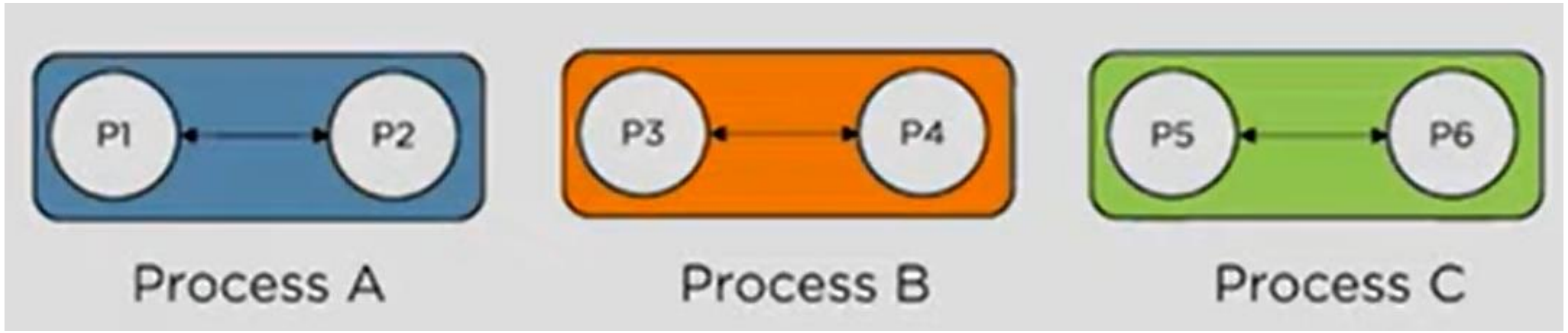
- You can create your own exceptions in Java. Keep the following points in mind when writing your own exception classes −
  - All exceptions must be a child of Throwable.
  - If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class.
  - If you want to write a runtime exception, you need to extend the RuntimeException class.

Difference between final, finally and finalize

Final	Finally	finalize
Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.
Final is a keyword.	Finally is a block.	Finalize is a method.

Thread Overview

- A thread is an individual light-weight, and a smallest unit of a given process. There are multiple threads in a single process and each thread is independent of the other.
- The Process of using the entire CPU of the computer and being capable to execute more than two processes or computational jobs at a time is defined as **multiprocessing**.



- A thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Thread Overview

- Java provides built-in support for multithreaded programming.
- A multithreaded program contains two or more parts that can run **concurrently**. Each part of such a program is called a thread and each thread defines a separate path of the execution. Thus, multithreading is a specialized form of multitasking.
- Threads reduce inefficiency by preventing the waste of CPU cycles.
- Thread and Thread subclass objects are not threads. Instead, they describe a thread's attributes, such as its name, and contain code (via a run() method) that the thread executes. When the time comes for a new thread to execute run(), another thread calls the Thread's or its subclass object's start() method.
- The JVM gives each thread its own method-call stack.

How to Create a Thread

- There are two ways to create a thread:
  - By extending Thread class
  - By implementing Runnable interface.

Thread class:

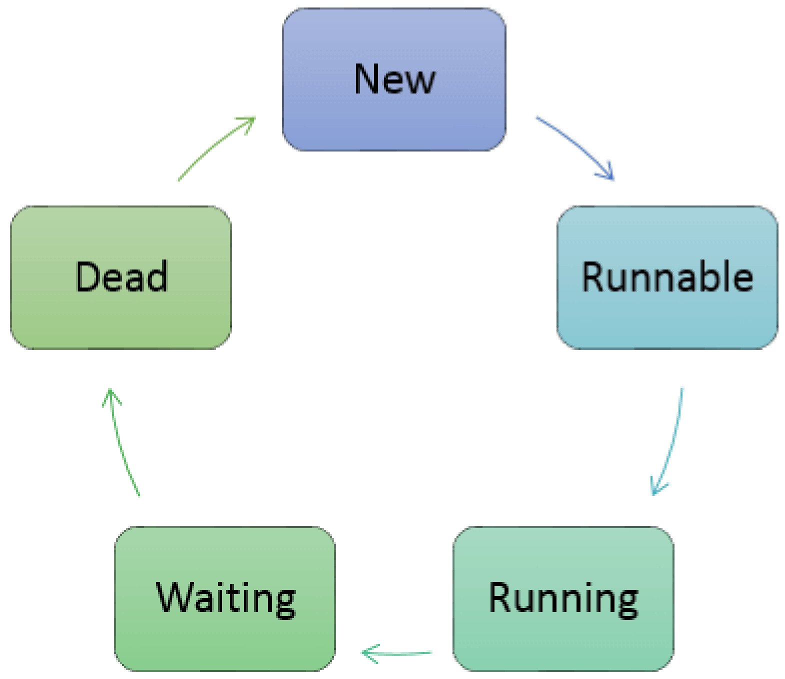
- Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Thread Life Cycle

- Threads exist in several states:
  - **New** - When we create an instance of Thread class, a thread is in a new state.
  - **Runnable**: The instance of the thread is invoked with a start method. The thread control is given to scheduler to finish the execution. It depends on the scheduler, whether to run the thread.
  - **Running** - The Java thread is in running state.
  - **Non-Runnable**: This is the state when a thread has to wait. As there multiple threads are running in the application, there is a need for synchronization between threads. Hence, one thread has to wait, till the other thread gets executed. Therefore, this state is referred as waiting state.
  - **Terminated (Dead)** - A thread can be terminated, which halts its execution immediately at any given time. Once a thread is terminated, it cannot be resumed.



Runnable Interface

- The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

    public void run(): is used to perform action for a thread.

Starting a thread:

- start() method of Thread class is used to start a newly created thread. It performs following tasks:
    - A new thread starts(with new callstack).
    - The thread moves from New state to the Runnable state.
    - When the thread gets a chance to execute, its target run() method will run.
- If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute.

Thread Class Methods

- Thread Names - When you create a Java thread you can give it a name. The name can help you distinguish different threads from each other
- The Thread.currentThread() method returns a reference to the Thread instance executing currentThread() . This way you can get access to the Java Thread object representing the thread executing a given block of code
- A thread can pause itself by calling the static method Thread.sleep() . The sleep() takes a number of milliseconds as parameter
- We can prevent the execution of a thread by using one of the following methods of Thread class.
  - The yield() method of thread class causes the currently executing thread object to temporarily pause and allow other threads to execute. yield() basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.

Thread Class Methods

- A yield() method is a static method of Thread class and it can stop the currently executing thread and will give a chance to other waiting threads of the same priority. If in case there are no waiting threads or if all the waiting threads have low priority then the same thread will continue its execution. The advantage of yield() method is to get a chance to execute other waiting threads so if our current thread takes more time to execute and allocate processor to other threads.

- Java Thread join method can be used to pause the current thread execution until unless the specified thread is dead.

Thread Priority

Priority of a Thread

- Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

- **3 constants defined in Thread class:**
  - public static int MIN_PRIORITY
  - public static int NORM_PRIORITY
  - public static int MAX_PRIORITY

- Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Thread Scheduler

- **Thread scheduler** in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Difference between preemptive scheduling and time slicing

- Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.
- Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

Threads

Sleep

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
- The Thread class provides two methods for sleeping a thread:
  - public static void sleep(long miliseconds)throws InterruptedException
  - public static void sleep(long miliseconds, int nanos)throws InterruptedException
    - millis − This is the length of time to sleep in milliseconds.
    - nanos − This is 0-999999 additional nanoseconds to sleep.

- **What if we call run method directly**
  - Normally each thread starts in a separate call stack.
  - Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

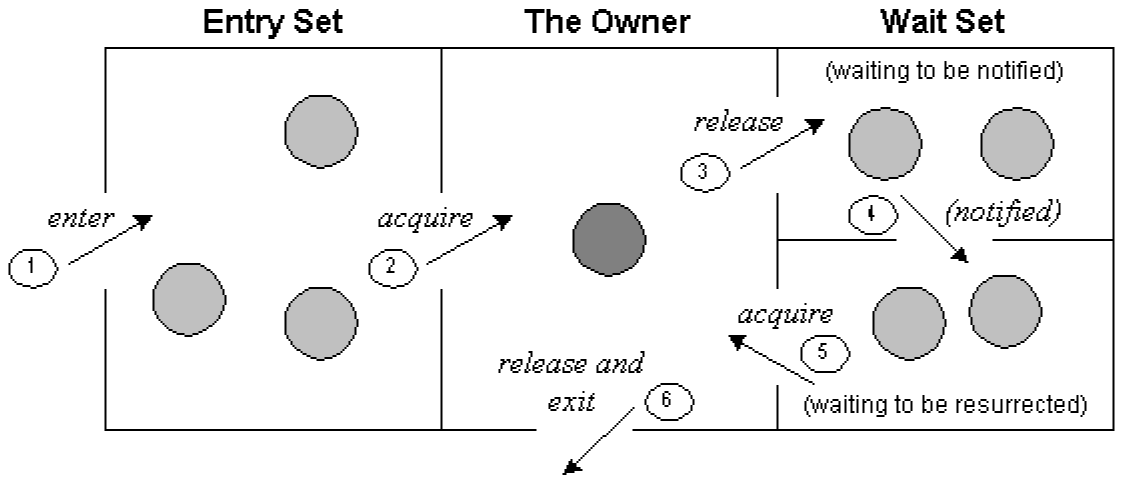
Multi Threading

Thread Synchronization

- In multithreading, there is the asynchronous behavior of the programs. If one thread is writing some data and another thread which is reading data at the same time, might create inconsistency in the application.
- When there is a need to access the shared resources by two or more threads, then synchronization approach is utilized.
- In this approach, once the thread reaches inside the synchronized block, then no other thread can call that method on the same object. All threads have to wait till that thread finishes the synchronized block and comes out of that.

Inter Thread Communication

- Inter-thread communication is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of Object class:
  - **wait()** - Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
  - **notify()** - Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.
  - **notifyAll()** - Wakes up all threads that are waiting on this object's monitor.



Thread Pools

- Thread Pools are useful when you need to limit the number of threads running in your application at the same time.
- Instead of starting a new thread for every task to execute concurrently, the task can be passed to a thread pool. As soon as the pool has any idle threads the task is assigned to one of them and executed.
- Java 5 comes with built in thread pools in the java.util.concurrent package, so you don't have to implement your own thread pool.
- Risks in using Thread Pools
  - Deadlock : While deadlock can occur in any multi-threaded program, thread pools introduce another case of deadlock, one in which all the executing threads are waiting for the results from the blocked threads waiting in the queue due to the unavailability of threads for execution.
  - Thread Leakage :Thread Leakage occurs if a thread is removed from the pool to execute a task but not returned to it when the task completed.
  - Resource Thrashing :If the thread pool size is very large then time is wasted in context switching between threads. Having more threads than the optimal number may cause starvation problem leading to resource thrashing as explained.

Q & A