CHANDAN MUKHERJEE

MTech (IT), BE (Computer Science)
SCJP (Java), Oracle (SQL) GLOBAL CERTIFIED
Microsoft Certified Innovative Educator
Corporate Trainer
chan.muk@gmail.com

JAVA
Abstract Class

Abstract Class Features

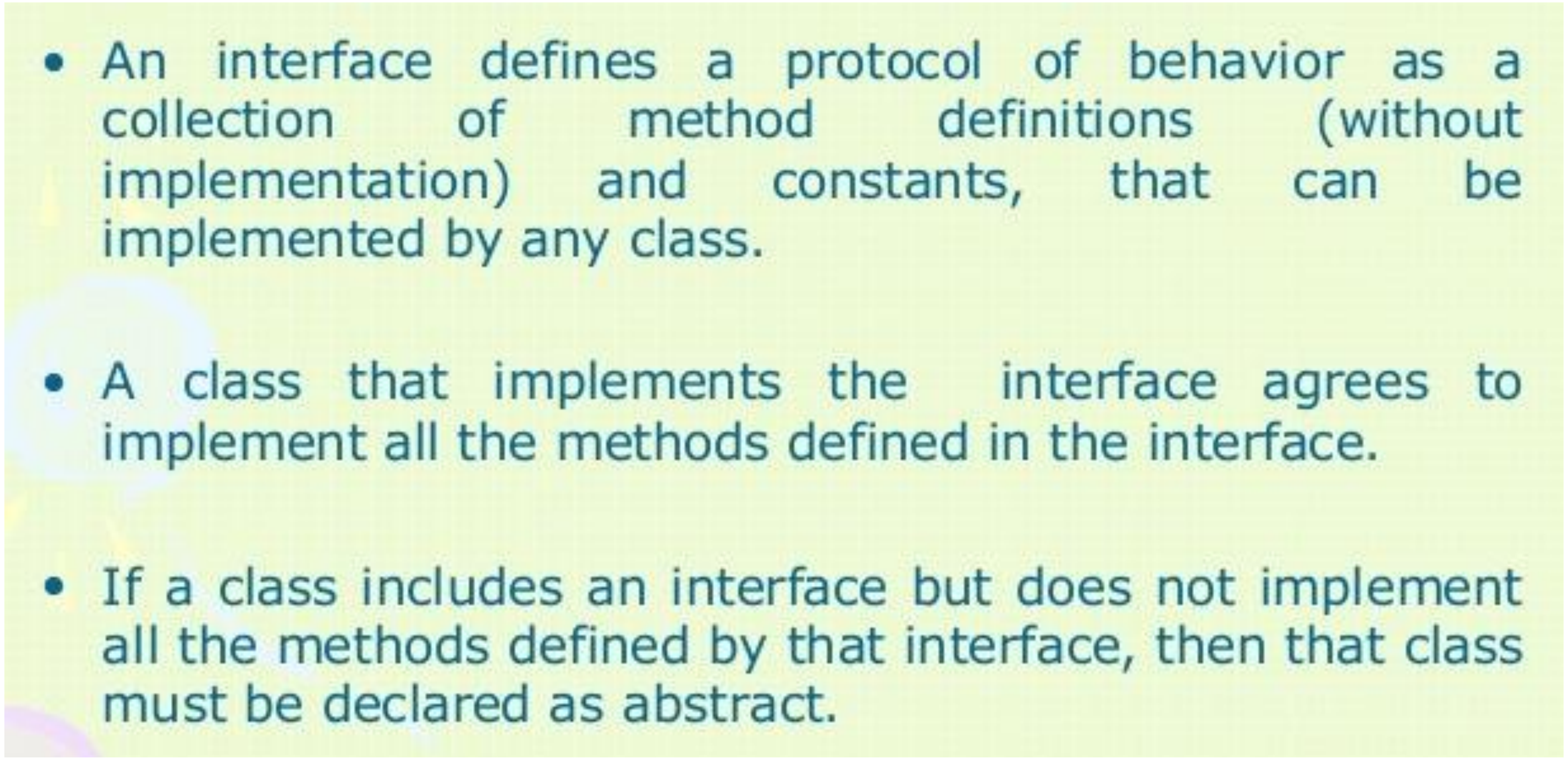
- Method with no body is call abstract method.
- If minimum one abstruct method present within a class then the class should be an abstract class.
- abstract keyword should be written before class name and method name
- abstract class contains normal method or concrete method also.
- We can not create object of an abstract class but reference can be created.
- To access the normal method of an abstract class a child class should be created and must override all abstruct method of the parent. (if not then child class became an abstract class)
- Using child class object we can access a normal method of an abstract class.

- PROGRAM

Can we create constructer within an abstruct class?

JAVA Interface

Interface



- An interface defines a protocol of behavior as a collection of method definitions (without implementation) and constants, that can be implemented by any class.
- A class that implements the interface agrees to implement all the methods defined in the interface.
- If a class includes an interface but does not implement all the methods defined by that interface, then that class must be declared as abstract.

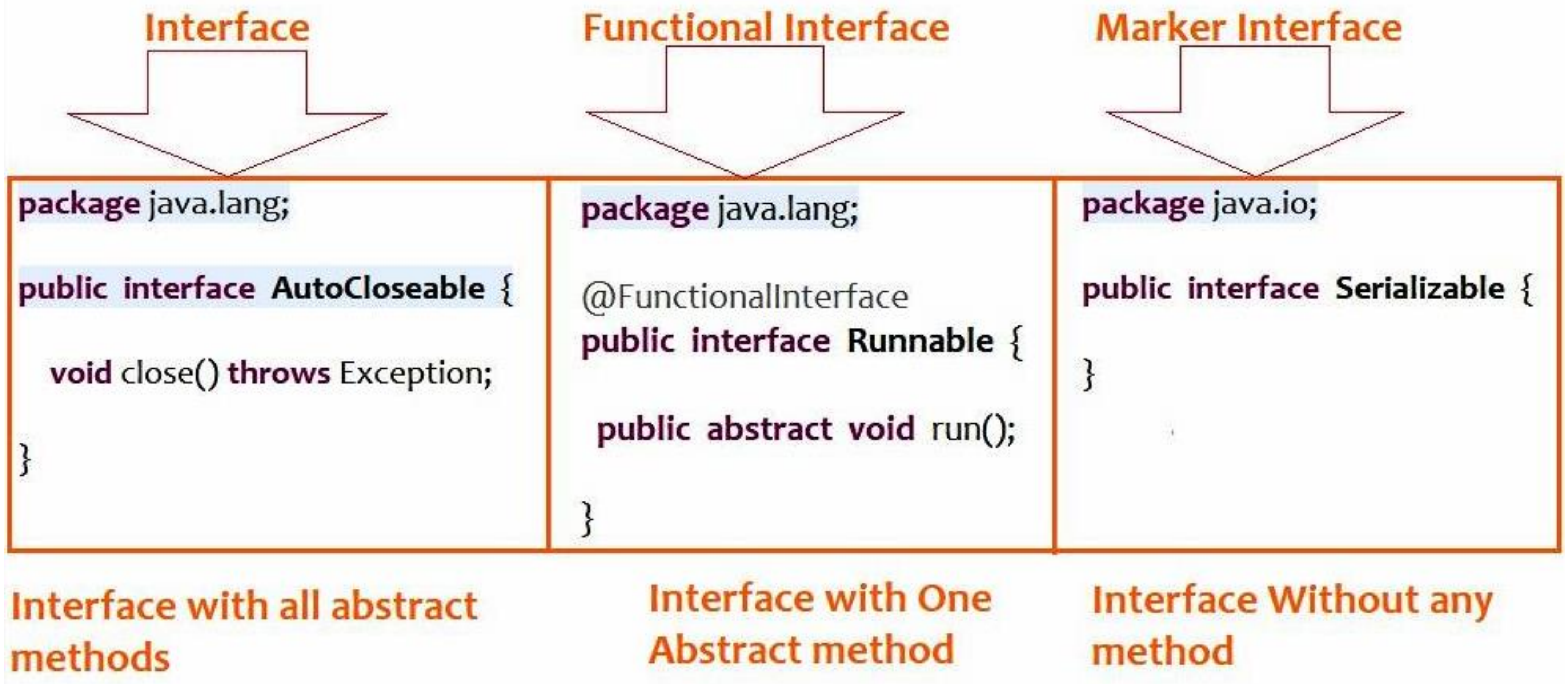
(Normal, Functional, Marker

JAVA INTERFACE NAMING CONVENTION

Interface Name - Interface name should start with an uppercase letter and be an adjective.

- PROGRAM

Types Of Interfaces In Java



Interface

```java
package java.lang;

public interface AutoCloseable {

   void close() throws Exception;

}
```

Interface with all abstract methods

Functional Interface

```java
package java.lang;

@FunctionalInterface
public interface Runnable {

  public abstract void run();

}
```

Interface with One Abstract method

Marker Interface

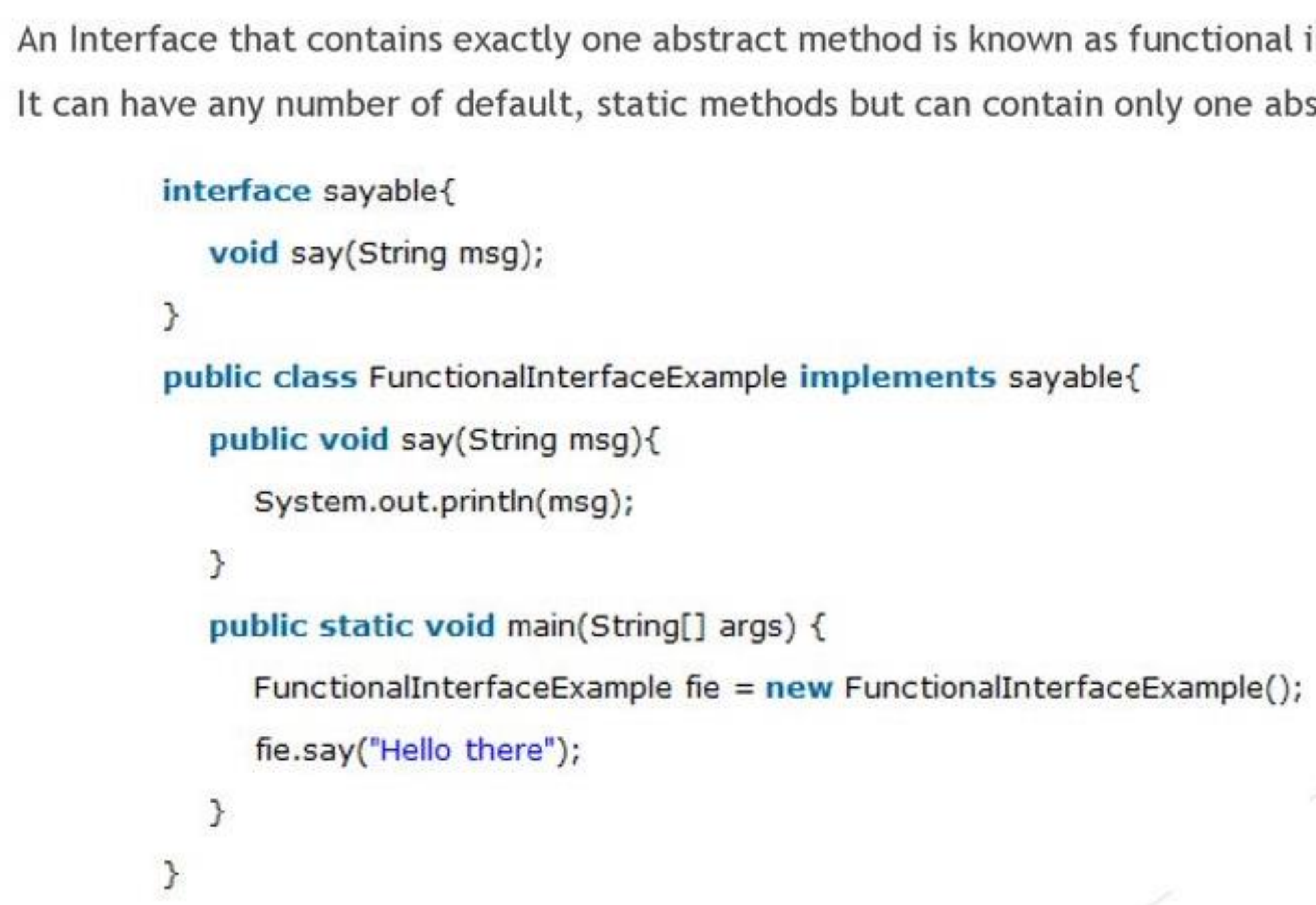
```java
package java.io;

public interface Serializable {

}
```

Interface Without any method

Functional Interface



An Interface that contains exactly one abstract method is known as functional interface.

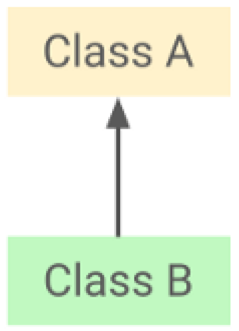
It can have any number of default, static methods but can contain only one abstract method.

```java
interface sayable{
    void say(String msg);
}
public class FunctionalInterfaceExample implements sayable{
    public void say(String msg){
        System.out.println(msg);
    }
    public static void main(String[] args) {
        FunctionalInterfaceExample fie = new FunctionalInterfaceExample();
        fie.say("Hello there");
    }
}
```

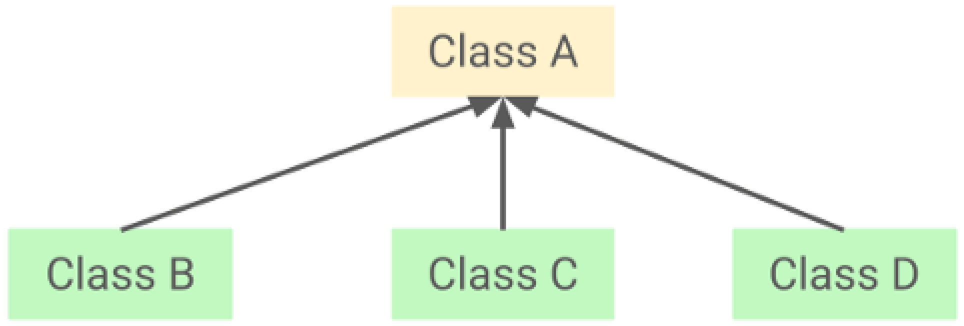
Marker Interface Examples

1. Serializable
2. Cloneable
3. Remote

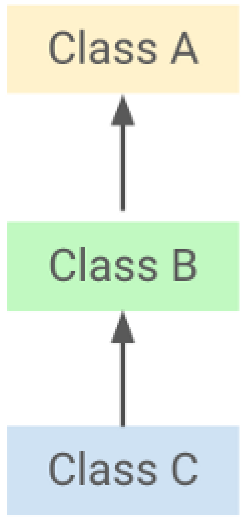
Interface Inheritance



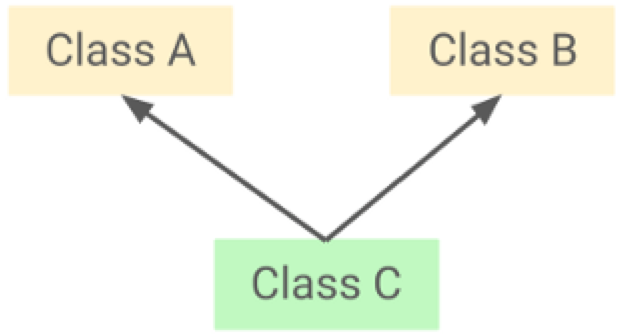
Single Inheritance



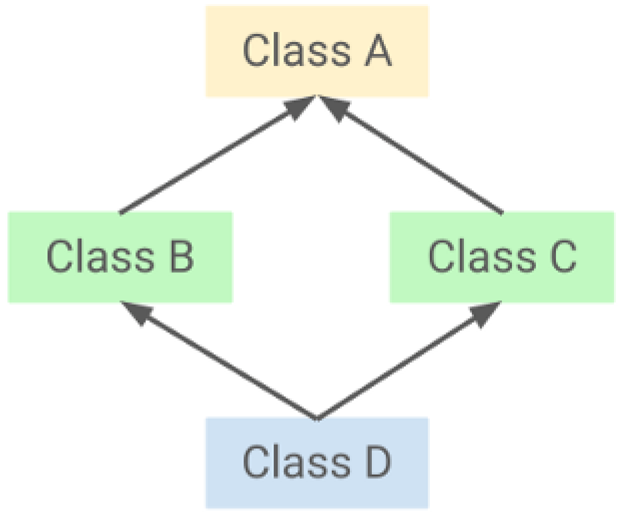
Hierarchical inheritance



Multilevel Inheritance



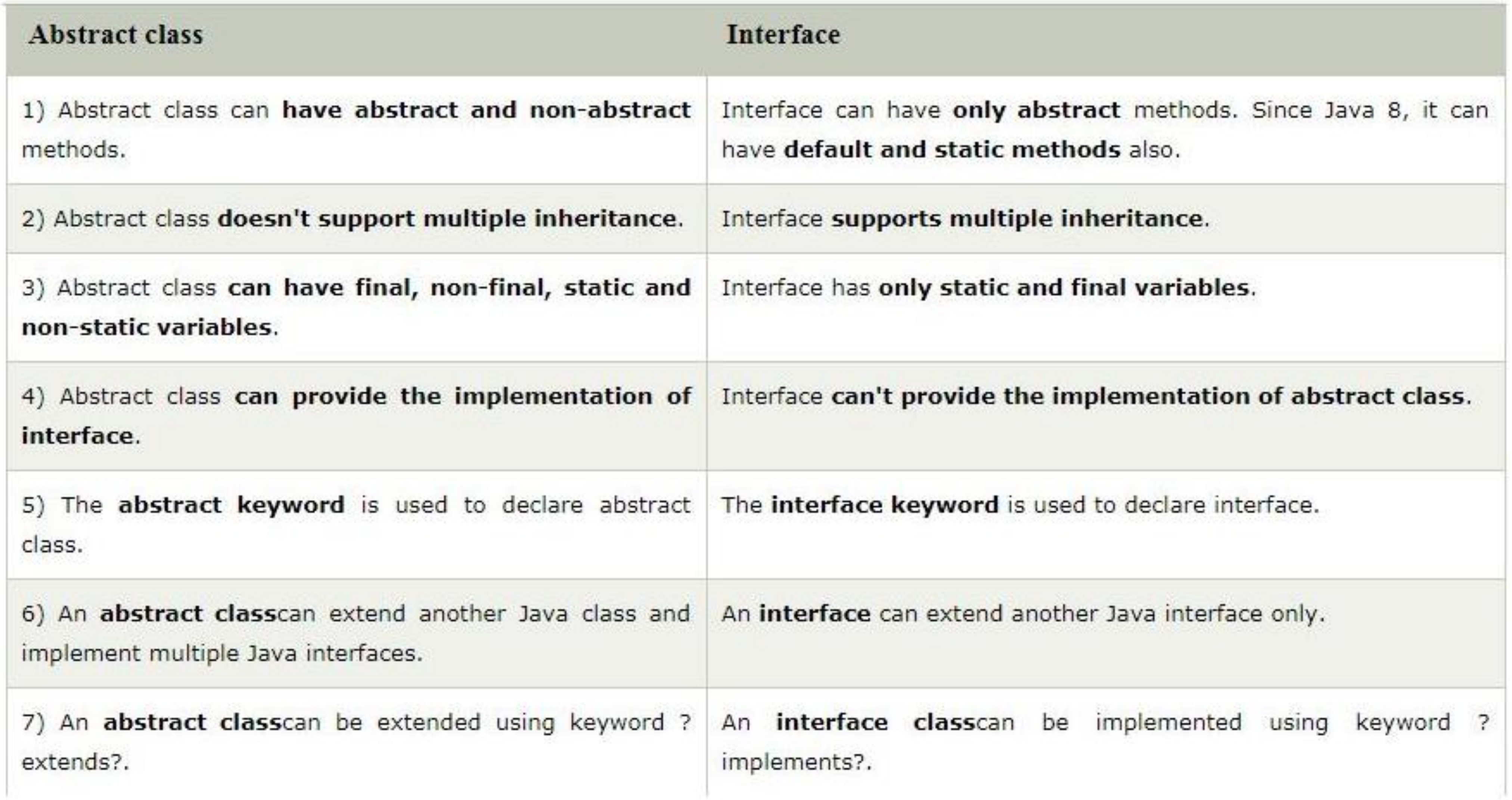
Multiple Inheritance



Hybrid Inheritance

Java

The 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 class**can be implemented using keyword ? implements?.

Lambda Syntax

- No arguments: `() -> System.out.println("Hello")`
- One argument: `s -> System.out.println(s)`
- Two arguments: `(x, y) -> x + y`
- With explicit argument types:

```
(Integer x, Integer y) -> x + y
```

- Multiple statements:

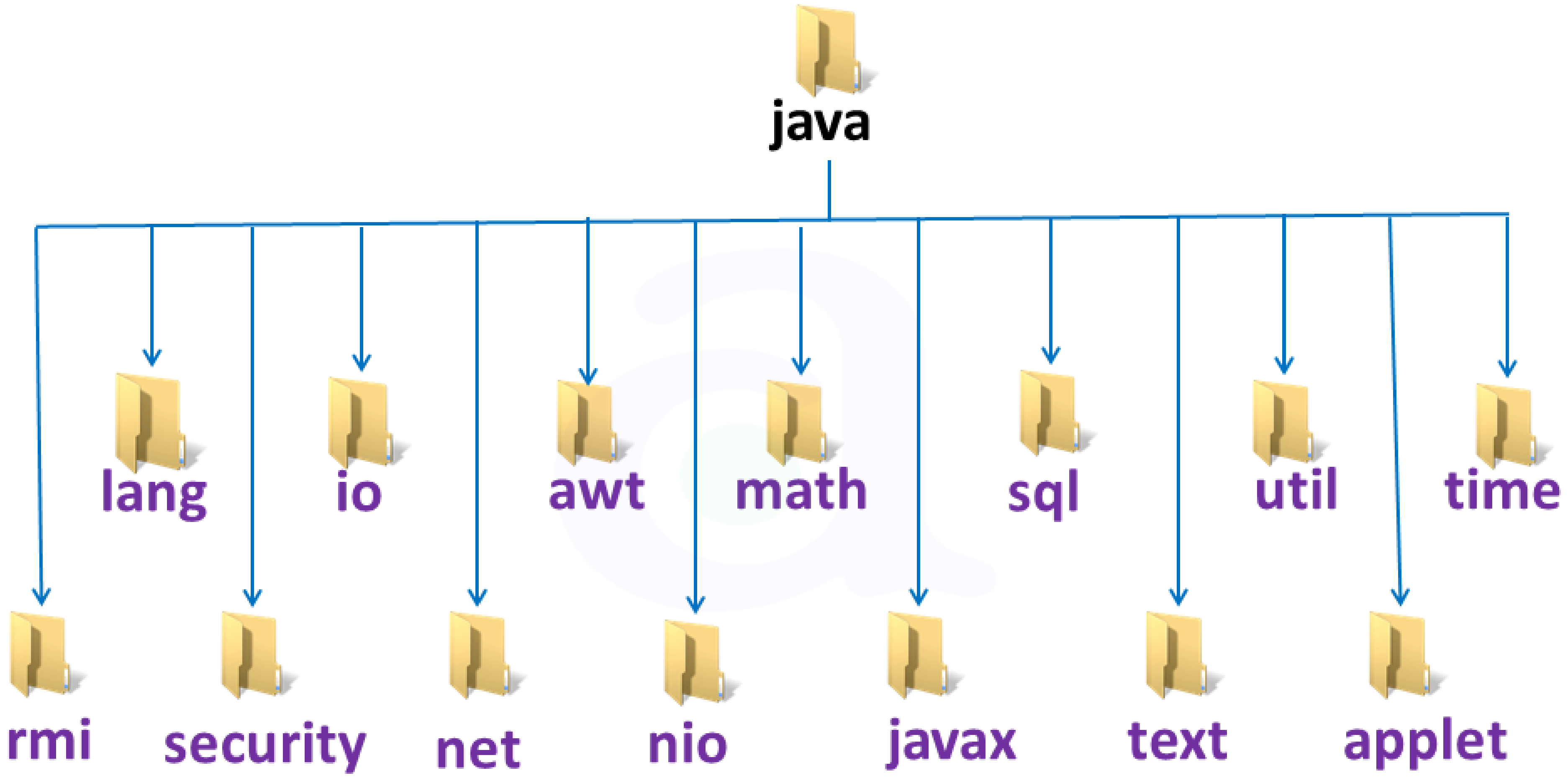
```
(x, y) -> {
    System.out.println(x);
    System.out.println(y);
    return (x + y);
}
```

- PROGRAM

JAVA Package

Package in Java

- ❑ Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.
- ❑ A package is a collection of related Java entities (such as classes, interfaces, exceptions, errors and enums).
- ❑ A **package** provides a mechanism for grouping a variety of similar types of classes, interfaces and sub-packages.
- ❑ Grouping is based on functionality.
- ❑ Java packages can be stored in compressed files called JAR files (Java Archieve)


java
lang
io
awt
math
sql
util
time
rmi
security
net
nio
javax
text
applet

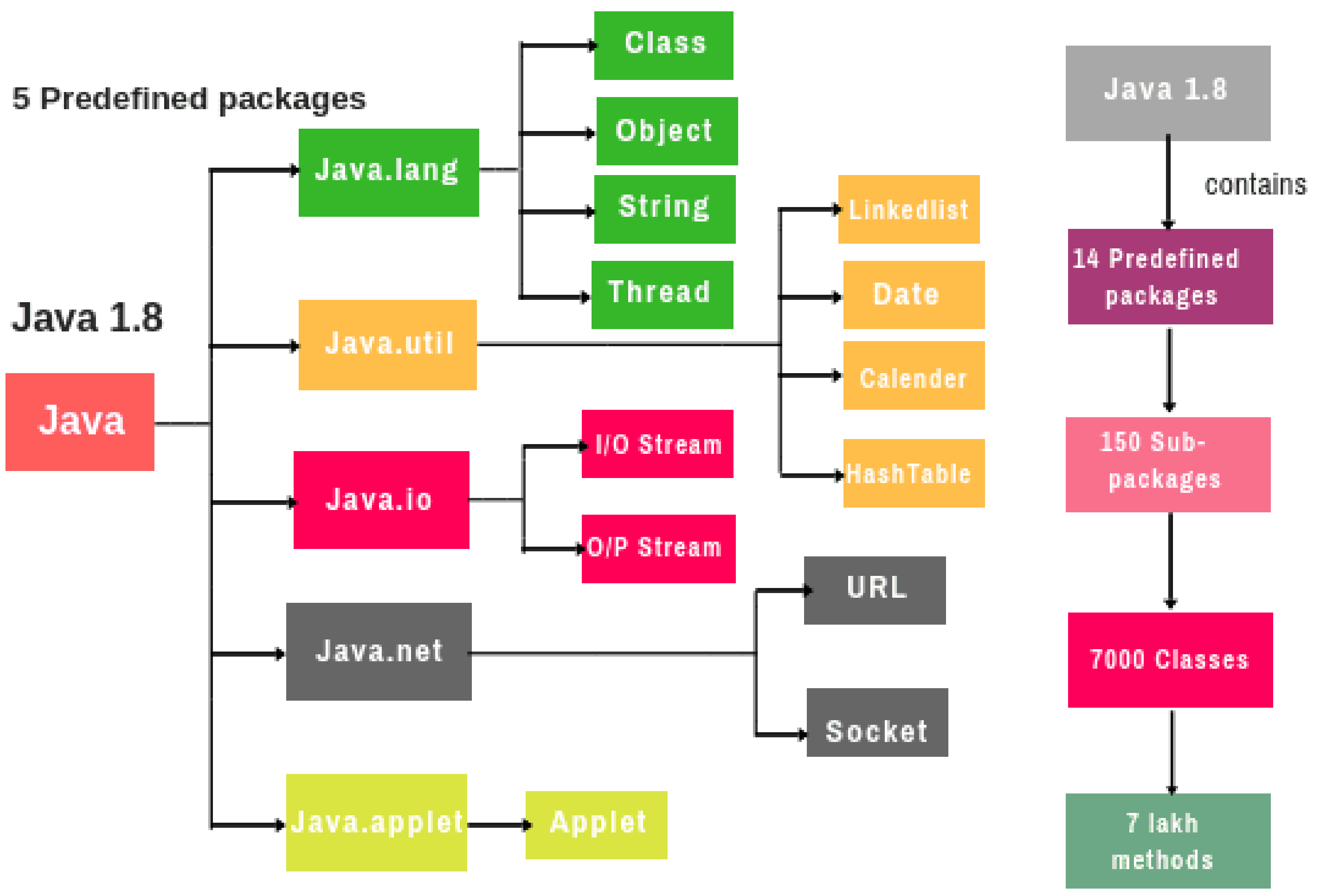
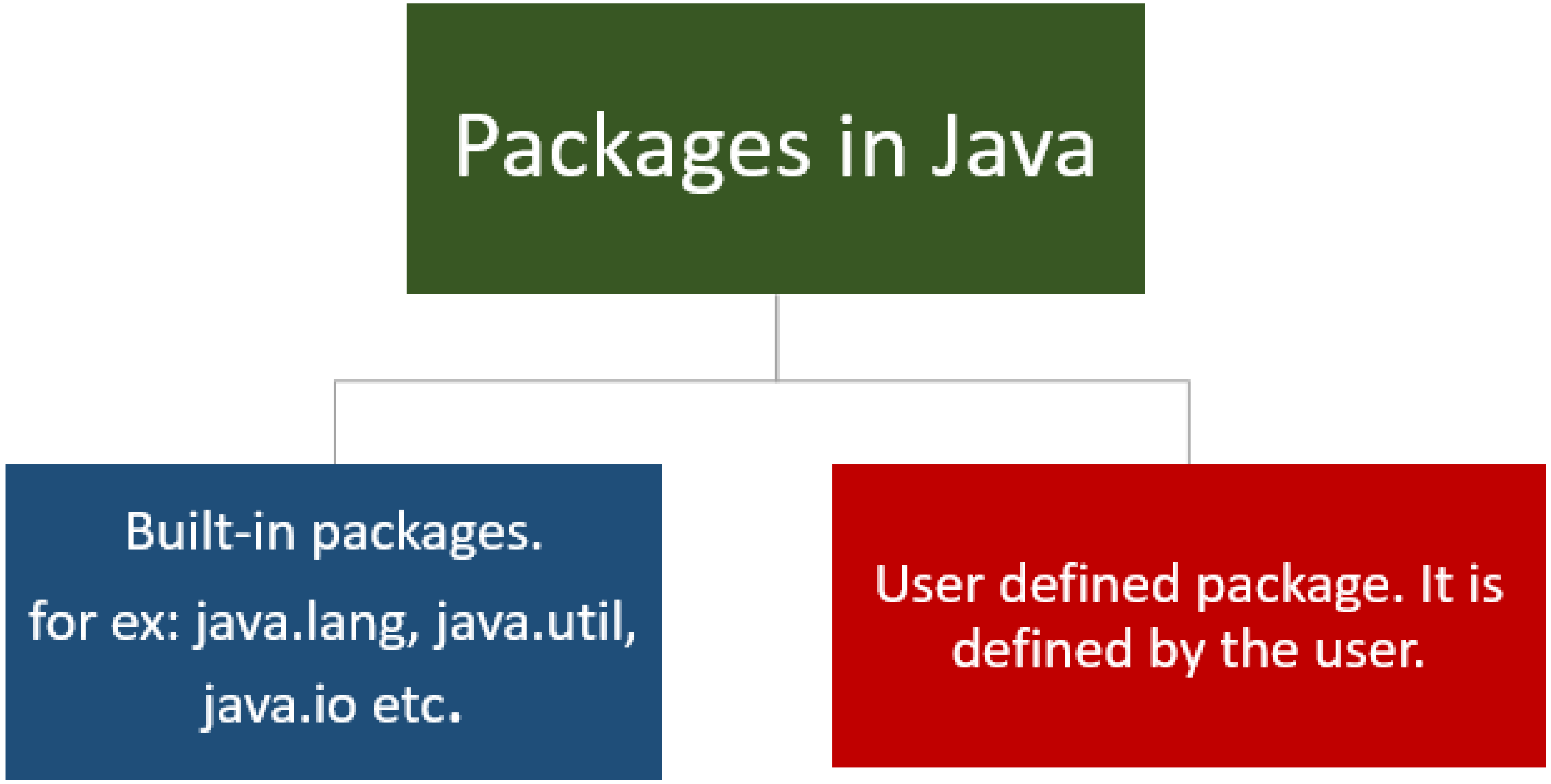
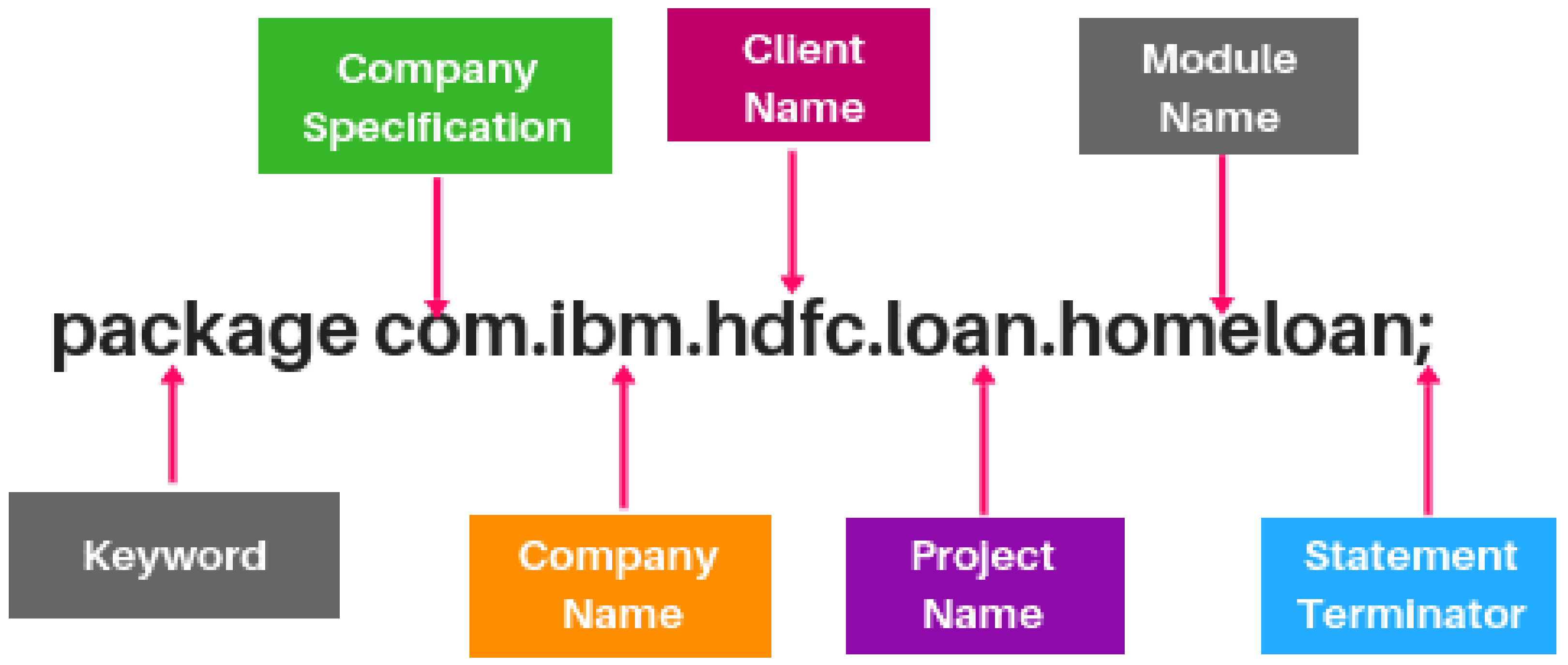


Fig: Predefined packages in Java

JAVA PACKAGE NAMING CONVENTION

Package Name - A package should be named in lowercase characters.


Packages in Java
Built-in packages.
for ex: java.lang, java.util, java.io etc.
User defined package. It is defined by the user.



While declaring the package name, every character should be lowercase

Fig: Complete Package Structure of Project

JAVA

Access Specifier

Java

Access Specifiers in Java

		public	private	protected	default
Same Package	Class	YES	YES	YES	YES
	Sub class	YES	NO	YES	YES
	Non sub class	YES	NO	YES	YES
Different Package	Sub class	YES	NO	YES	NO
	Non sub class	YES	NO	NO	NO

- ✓private - Private member can be access only within the same class.

- ✓default - Default member can be access within same class & within the same package.

- ✓protected - Protected member can be access within same class, within same package & from child class present in different package.

- ✓public - Public member can be access within same class, within same package & from different package.

- PROGRAM

JAVA
Exception Handling

Exception Handling

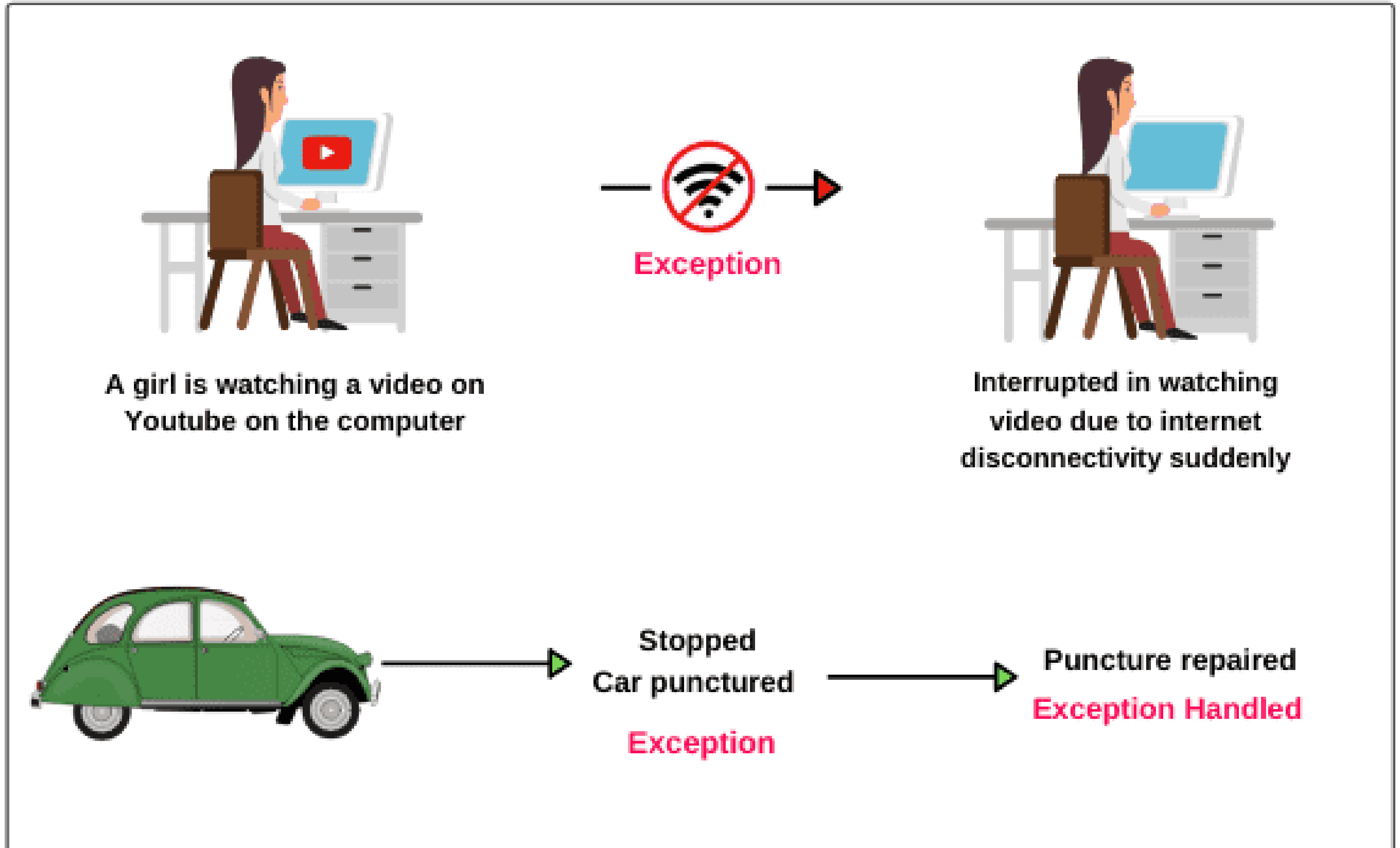
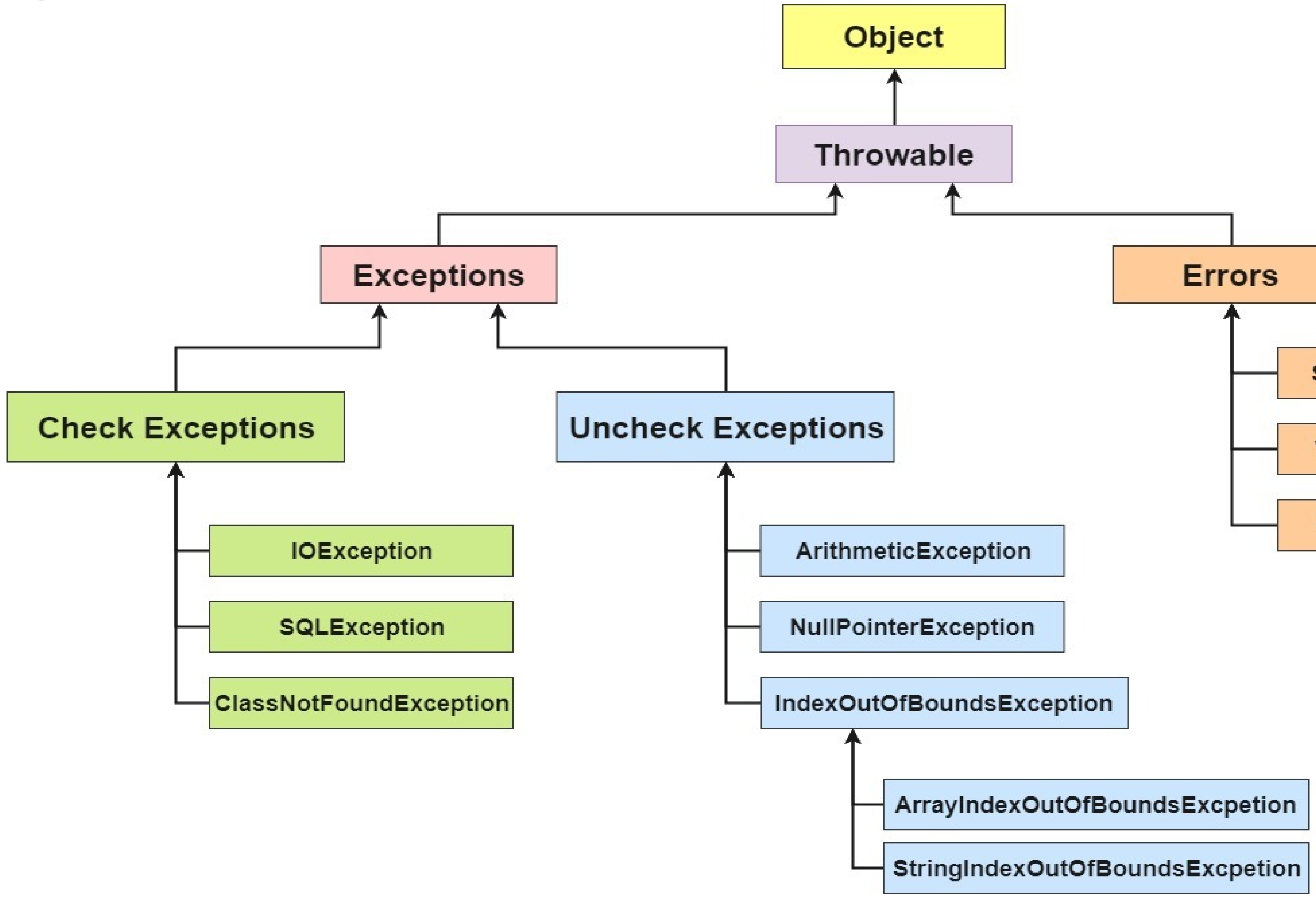

A girl is watching a video on Youtube on the computer
Exception
Interrupted in watching video due to internet disconnectivity suddenly
Stopped
Car punctured
Exception
Puncture repaired
Exception Handled

Fig: Realtime Example of Exception Handling


Object
Throwable
Exceptions
Errors
Check Exceptions
Uncheck Exceptions
StackOverFlowError
VirtualMachineError
OutOfMemoryError
IOException
SQLException
ClassNotFoundException
ArithmeticException
NullPointerException
IndexOutOfBoundsException
ArrayIndexOutOfBoundsExcpetion
StringIndexOutOfBoundsExcpetion

Exception Handling in Java

Try

Catch

Throw

Throws

Finally

- PROGRAM

Errors	Exception
A lack of system resources typically causes errors in a program, and the program that a programmer writes is not designed to detect such issues.	An exception occurs mainly due to issues in programming such as bugs, typos, syntax errors etc.
Errors usually happen at runtime. Therefore they're of the unchecked type.	Exceptions can arise at both runtime and compile time.
System crashes and out of memory errors are two instances of errors.	SQLException is an example of exceptions in Java
Errors belong to java.lang.error.	Errors belong to java.lang.Exception.
Errors are irrecoverable and lead to abnormal termination of the program.	Exceptions are recoverable and can be handled by exception handling techniques.

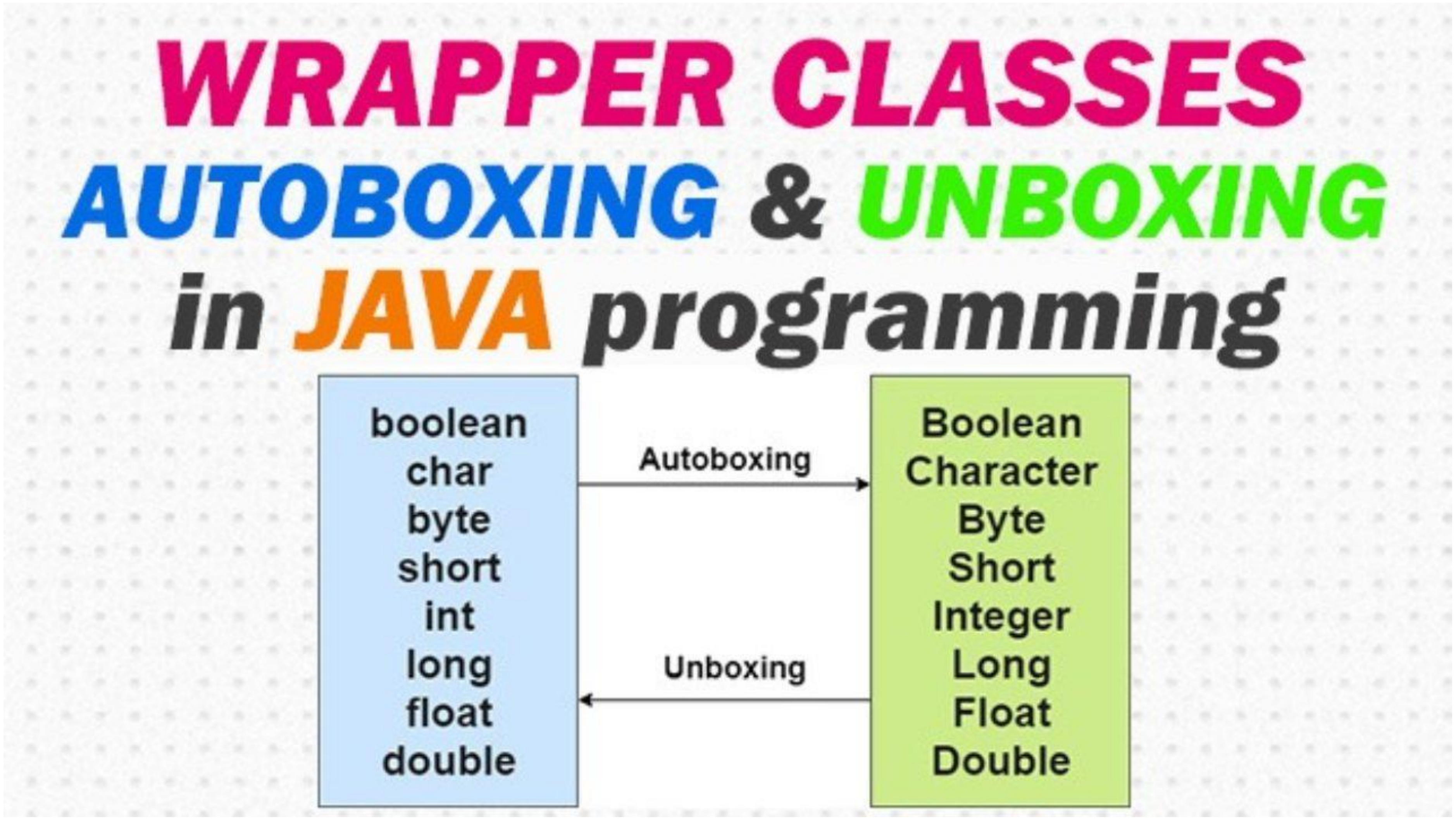
JAVA

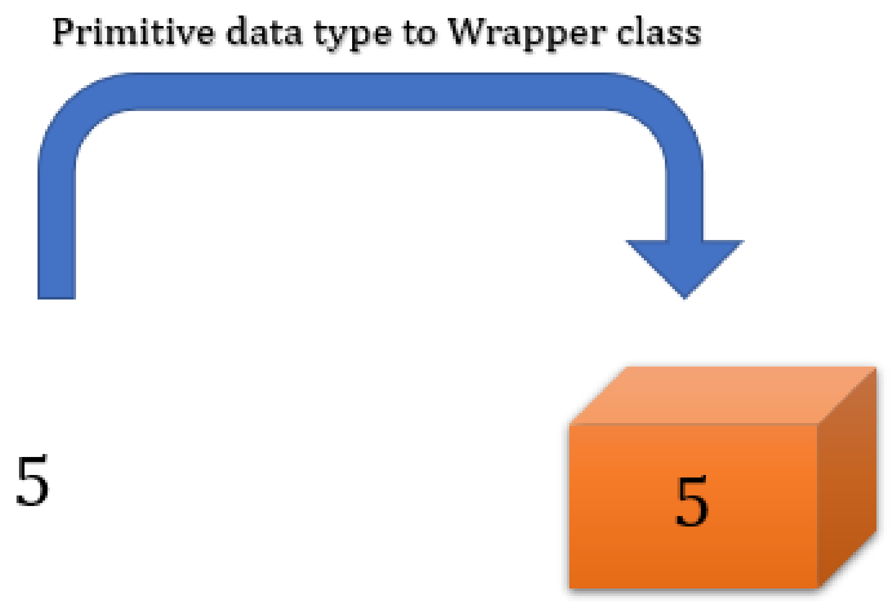
Wrapper Class

Wrapper Class in Java

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

Since **J2SE 5.0**, auto-boxing and unboxing features convert primitives into objects and objects into primitives automatically. The automatic conversion of a primitive into an object is known as auto-boxing and vice-versa unboxing.


WRAPPER CLASSES
AUTOBOXING & UNBOXING
in JAVA programming
boolean
char
byte
short
int
long
float
double
Autoboxing
Unboxing
Boolean
Character
Byte
Short
Integer
Long
Float
Double



int a = 5

Integer a = 5

- PROGRAM

JAVA Generics

Java generics

- lets you write code that is safer and easier to read
- is especially useful for general data structures, such as ArrayList

- generic programming = programming with classes and methods parameterized with types

JAVA

Collection Framework

- Collection Framework => interfaces + classes

- ALL Collection Framework classes and interfaces are by default generic from v 1.5.
- So **it will only works with WRAPPER CLASS** or **any user defined classes**.

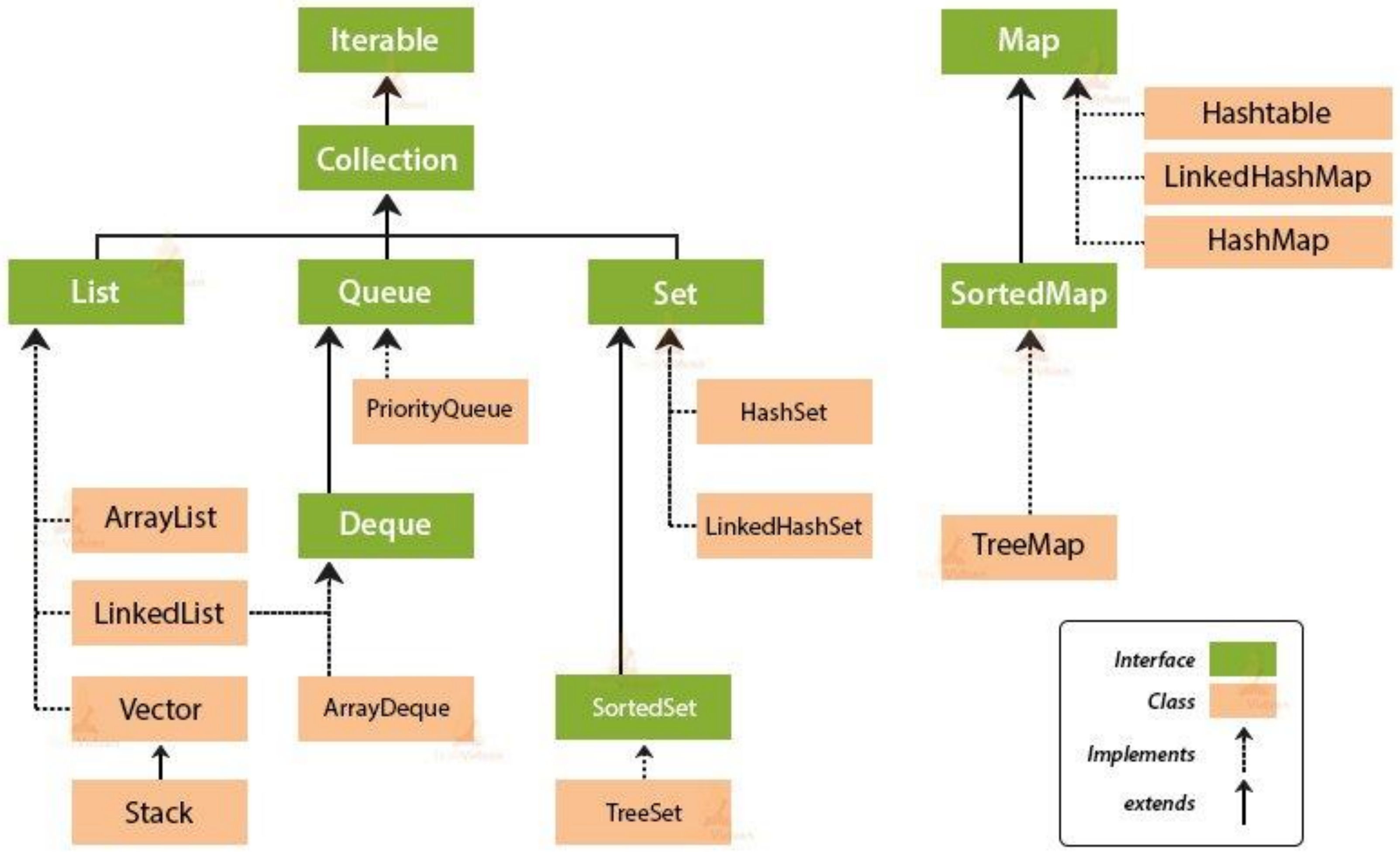
- Primitive datatype is not supported by Generic (Collection Framework)

Advantages of Collection Framework

- ✓ To write data structure independent code
- ✓ Efficient Memory Management
- ✓ lots of System defined methods are available to do the jobs.

Feature	Description
Performance	The collection framework provides highly effective and efficient data structures that result in enhancing the speed and accuracy of a program.
Maintainability	The code developed with the collection framework is easy to maintain as it supports data consistency and interoperability within the implementation.
Reusability	The classes in Collection Framework can effortlessly mix with other types which results in increasing the code reusability.
Extensibility	The Collection Framework in Java allows the developers to customize the primitive collection types as per their requirements.

Collection Framework Hierarchy in Java



Comparable	**Comparator**
1) Comparable provides a **single sorting sequence** with natural ordering. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences** for different attributes. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable **affects the original class**, i.e., the actual class is modified.	Comparator **doesn't affect the original class**, i.e., the actual class is not modified.
3) Comparable provides **compareTo(Object a) method** to sort elements. Comparable interface compares “this” reference with the object specified.	Comparator provides **compare(Object o1, Object o2) method** to sort elements. Comparator in Java compares two different class objects provided.
4) Comparable interface belongs to **java.lang** package.	Comparator interface belongs to **java.util** package.
5) We can sort the list elements of Comparable type by **Collections.sort(List)** method.	We can sort the list elements of Comparator type by **Collections.sort(List, Comparator)** method.