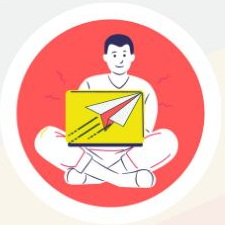
# Java mastery

# Generics in java

1. Introduced in java for two main purpose:
   a. For Type Safety
   b. Avoid Manual Type  Casting
2. They enable us to create classes, interfaces, and methods that can seamlessly work with various data types without sacrificing compile-time safety.
3. Introduced in java 1.5 ( java 5)

# Wildcard in Java Generics



1. There are three types of wildcard :
    a. Upperbound Wildcards **<? extends Type>**
    b. Lowerbound Wildcards **<? super Type>**
    c. Unbound **<?>**

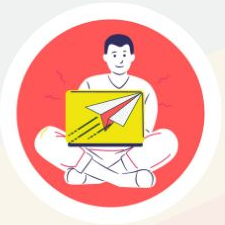
# Java Inner Classes

1. Class created inside another class is called inner class.
2. Inner classes is used  to group logical related classes together at one place make more readable.
3. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.

# Inner class Types

1. Non-static nested class (inner class)
    i. Member inner class/Regular Inner class
    ii. Anonymous inner class
    iii. Local inner class / Method local inner classes
2. Static nested class
3. Nested Interface

# Regular Expression



1. Regex is a powerful tool in Java for defining patterns that can be used for searching, manipulating, and editing strings.
2. **java.util.regex** package defined classes to work with regex in java.
3. **Pattern** Class- Represent Compile version of regex
4. **Matcher** Class- Represent regex engine perform maches
5. **PatternSyntaxException**
6. **PatternResult** interface

# Use of Regex

1. Validating user input (e.g., email addresses, passwords).
2. Extracting specific information from text (e.g., phone numbers, dates).
3. Searching and replacing text based on patterns.

# Regular Expression PP

1. Create RE that accept alphanumeric characters only.
2. Create RE that accept 10 digits number only.
3. Write a EX to match email addresses.
4. Write a RE for matching username that contain numbers, letters and @,$ only.

# Java 8 new features

18th March 2014

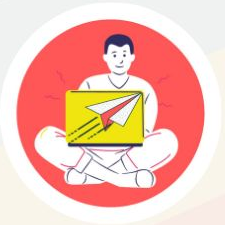
1. Lambda expressions
2. Functional interfaces
3. Method references
4. Stream API
5. Default methods
6. Static methods in interfaces
7. Optional Class
8. Collector class
9. forEach methods
10. JDBC Enhancements et

# Lambda Expression

1. A lambda expression is a compact way to represent an anonymous function (a function without a name)
2. More concise way to implement Functional interfaces (Interface having single method)
3. Lambda expressions were introduced in Java 8 to bring functional programming concepts to the Java language.

```
Runnable runnable = () -> {
        // code goes here
    };
```

# Why lambda



1. More concise and readable code.
2. Support functional programming
3. Enhance api design
4. More performance

# Functional Interfaces

1. Interfaces that contain only **one abstract** method is called functional interfaces.
2. Helps to achieve Functional programming

```java
@FunctionalInterface
public interface Runnable
{
    /**
     * Runs this operation.
     */
    void run();
}
```

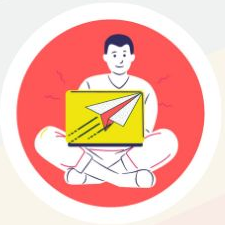
# Method References

1. Method References allows you to refer to methods or constructors without invoking them directly.
2. Static method references
3. Not-static method references
4. Constructor references

# Stream API

1. Introduced in Java 8, Stream API is used to process collections of objects.
2. A stream in Java is a sequence of elements that supports various methods which can be pipelined to produce the desired result.
3. Stream does not store elements.
4. Stream is lazy and evaluates code only when required.
5. Pipelining
6. Parallelism: Streams can leverage parallel processing to improve performance on multi-core processors.
7. Immutable Data:  They do not modify the underlying data source.
8. The elements of a stream are only visited once during the life of a stream.

# Commonly used operations



1. forEach
2. filter
3. map
4. reduce
5. collect
6. findFirst
7. sorted
8. distinct
9. flatMap
10. limit
11. skip

# Java Default Methods inside interfaces

1. From java 8 we can create default methods inside interfaces .
2. Method with **default** keyword

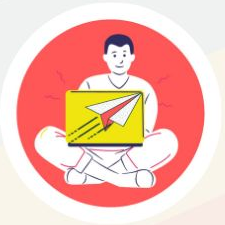
```
interface Findable{
    default void test(){
        System.out.println("default
method");
}
}
```

# Java static Methods inside interfaces

1. From java 8 we can create static methods inside interfaces .
2. Used to create utility methods

```
interface Findable{
    static void test(){
        System.out.println("default
method");
}
}
```

# forEach() method



1. New method is given in **Iterable** interfaces **forEach()** to iterate over elements.
2. From java 8(1.8)

# Java Collectors class

1. New java that provide implementation of different types of reduction methods.
2. java.util.steam.Collectors

# Java StringJoiner

1. String class use to create string with joining the string separated by delimiter.
2. We can create string by passing delimiters like comma(,), hyphen(-) etc

# Type Inference

1. In old version :
    a. List<**Integer**> list=new ArrayList<**Integer**>();

2. In java 8 :
    a. List<**Integer**> list=new ArrayList<>();

# Optional Class

1. Present in **java.util** package
2. Public final class use to deal with NullPointerException .
3. Use to represent optional value instead of **null** references.

# Type Annotation

1. We have power to declare annotations with type.
2. With declaring variables , we have use annotations.
    a. **@NonNull String str;**

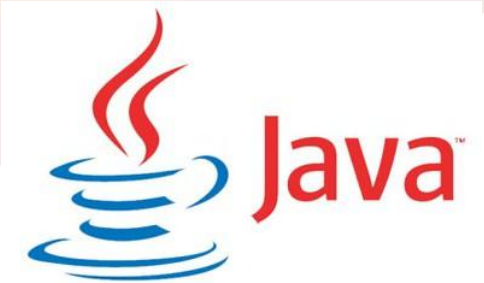
**Java created type annotations to support improved analysis of Java programs. It supports way of ensuring stronger type checking.**

# Java 9 Features

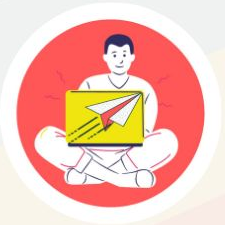
1. JPMS (Java Platform Module System) - Project Jigsaw
2. JShell- REPL
3. Improvement to Stream API
4. Private Interface methods
5. Try-With-Resource Improvement
6. Process API Updates
7. Diamond Operator Extension
8. @SafeVarargs Annotations
9. Collection Factory Methods
10. Multi-Resolution Image API

# JShell - REPL

1. The Java Shell tool (JShell) is an interactive tool for learning the Java programming language and prototyping Java code.
2. JShell is a Read-Evaluate-Print Loop tool (REPL), which evaluates declarations, statements, and expressions as they are entered and immediately shows the results.



# Java Module System

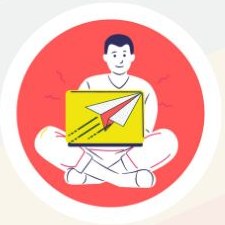


1. A Module is a group of closely related packages and resources along with a new module descriptor file.

# Module Descriptor

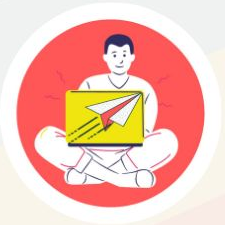
1. When we create a module, we include a descriptor file that defines several aspects of our new module:
2. **Name** – the name of our module
3. **Dependencies** – a list of other modules that this module depends on
4. **Public Packages** – a list of all packages we want accessible from outside the module
5. **Services Offered** – we can provide service implementations that can be consumed by other modules
6. **Services Consumed** – allows the current module to be a consumer of a service
7. **Reflection Permissions** – explicitly allows other classes to use reflection to access the private members of a package

# Improvement to Stream API



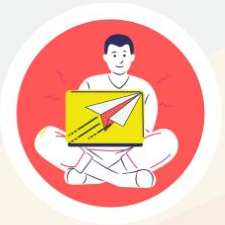
1. New methods is added to stream interface of java 9
2. takeWhile
3. dropWhile
4. OfNullable
5. iterate

# Private method in interface



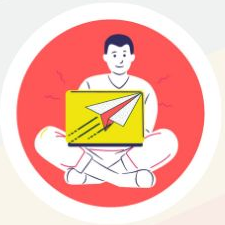
1. From java 9 we can create private method inside interface.

# Try with resource enhancement



1. Try with resource was introduced in java 7
2. Try with resources is enhanced in java 9
3. Lets understand with example

# Java 9 Inner class enhancement



1. From Java 9 allows use diamond operator in side anonymous classes.
2. Features is added in java 9 to add **Type Inference** .

# Collection Static Methods

1. Java 9 Collection library includes static factory methods for List, Set and Map interface.
2. These methods are useful to create small number of collection.
3. List.of(Elements..)
4. Set.of(Elements..)
5. Map.of(Elements..)