# Language Essentials

By Rahul Barve

# Objectives

- Inner Classes
- Using Enums
- Reflection API
- Lambda Expressions
- Date and Time API

# Introduction to Inner Classes

By Rahul Barve

# Introduction to Inner Classes

- In Java, it is possible to declare a class inside another class.
- Such a class is known as an inner class.

# Introduction to Inner Classes

```
public class OuterClass {
    …..
    public class InnerClass {
      ….
    }

}
```

# Why Inner Classes

By Rahul Barve

# Why Inner Classes

- Inner classes are useful when a complex logic is to be isolated but handled locally within the enclosing class.
- This enables enclosing classes to handle the operations without the overhead of method invocations, sending parameters and so on.

# Why Inner Classes

- Inner classes are heavily used in a GUI programming model for handling events.
- They are also used for UI component creation.

# Types of Inner Classes

By Rahul Barve

# Types of Inner Classes

- Inner classes are divided into 4 types:
  - Static Inner Class
  - Nested Class
  - Local Class
  - Anonymous Inner Class

# Static Inner Class

By Rahul Barve

# Static Inner Class

- An inner class declared with `static` modifier is called as a static inner class.
- It can be instantiated directly using the name of the outer class but can access only static members of an outer class.

# Static Inner Class

```
public class OuterClass {
    …..
    public static class InnerClass {
      ….
    }
  }
```

# Nested Class

By Rahul Barve

# Nested Class

- An inner class declared without `static` modifier is called as a nested class.
- It has to be instantiated always by using an object of an outer class but can access static as well as non-static members of an outer class.

# Nested Class

```
public class OuterClass {
    …..
    public class InnerClass {
      ….
    }
  }
```

# Local Class

# Local Class

- An inner class declared within a method's definition is known as a local class.
- It is loaded and instantiated for every method invocation.

# Local Class

```
public class OuterClass {
    public void myMethod(){
          class InnerClass {
               ….
          }
    }
}
```

# Anonymous Inner Class

By Rahul Barve

# Anonymous Inner Class

- An inner class declared without any name is an anonymous inner class.
- It is defined, loaded and instantiated within the invocation of a method or a constructor.
- It is used in the context of abstract classes or interfaces.

# Enums

By Rahul Barve

# Enums

- An enum type is a special data type that enables for a variable to be a set of predefined constants.
- The variable must be equal to one of the values that have been predefined for it.

# Enums

- Enums are declared using `enum` keyword.
- E.g.

```
public enum Nationality {
   INDIAN,   US,   GERMAN,   BRITISH,
   FRENCH, JAPANESE, OTHER
 }
```

# Enums

- Once an enum is declared, it can be used by using a `dot (.)` operator.
- E.g.

```
Nationality nt = Nationality.INDIAN;
```

# Enums

```
if(nt.equals(Nationality.INDIAN)){
    ….
  }
```

# Reflection

By Rahul Barve

# Reflection

- Sometimes, it's necessary to retrieve information about the class and perform some operations at runtime.
- Java provides a Reflection API that belongs to a package `java.lang.reflect`.

# Reflection API

By Rahul Barve

# Reflection API

- Reflection API mainly consists of 4 classes:
  - `java.lang.Class`
  - `java.lang.reflect.Method`
  - `java.lang.reflect.Constructor`
  - `java.lang.reflect.Field`

# Lambda Expressions

# Lambda Expressions

- A lambda expression is a new syntax element and operator into the Java language.
- The operator -> sometimes referred to as a *lambda operator* or an *arrow operator*.

# Lambda Expressions

- The lambda operator divides the expression into 2 parts.
- The left side indicates Lambda Parameters whereas the right side indicates Lambda Body.

# Lambda Expressions

- Lambda bodies are divided into 2 types:
  - Single Expression Lambda
  - Blocked Lambda

# Date and Time API

# Date and Time API

- With the release of JDK 8, Java now includes another approach to handing time and date.
- The Date and Time API of JDK 8 simplifies processing of date and time.

# Why Date and Time API

# Why Date and Time API

- The existing classes aren't thread-safe, leading to potential concurrency issues for users.
- Some of the date and time classes also exhibit quite poor API design.

# Why Date and Time API

- For example, years in `java.util.Date` start at 1900, months start at 1, and days start at 0—not very intuitive.

# Date and Time API

# Date and Time API

- The `java.time` package is the heart of Date / Time API.
- It mainly consists of 3 classes:
  - `LocalDate`
  - `LocalTime`
  - `LocalDateTime`

# Lets Summarize

- Inner Classes
- Using Enums
- Reflection API
- Lambda Expressions
- Date and Time API