ENUM , ANNOTATIONS, LAMBDAS, MODULES

ANNOTATIONS

Used for providing Compiler, Build-time and Runtime instructions.
Build tools may scan java code for annotations and generate source code or other files based on annotations.
Most of the annotations does not exist in compiled code.
Custom defined annotations can exist in runtime and are accessed via java reflections.

@deprecated (better to have java doc explaining why it is deprecated)
@override
@supresswarnings

Creating Annotations:
Annotations are defined in its own file just like classes or interfaces.

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnnotation { String Value() default ""; String name(); }

@MyAnnotation(name="Ravi")
public class MyClass{}
```

RetentionPolicy.CLASS . (Will be retained in .class but not in runtime)
RetentionPolicy.SOURCE (will not be avail in class; can be used for build tools)

@Target({ELEMENTTYPE.METHOD})
Used to specify which java elements your custom annotation can be used to annotate.

ELEMENTTYPE.{METHOD/ PACKAGE/ FIELD / CONSTRUCTOR / PARAMETER / TYPE}
TYPE above can be class, interface, enum or annotation.

@Inherited
This annotation signals that a custom java annotation used in a class should be inherited by subclasses inheriting from that class.

@Documented
Signals javadoc tool that annotation should be visible in the javadoc.

MODULES

Package Java application and packages into Java modules.
It specifies which packages should be visible to other modules using this module.
It also should specify which other modules are needed to run this module.
Also referred as Java JigSaw or Project JigSaw.
Helps reduce the size of the distribution jar.

ENUM

Special Java class

```
public enum Level {
  HIGH, MEDIUM, LOW
}

Level level  = Level.HIGH;
```

Enum Iteration
```
for(Level level: Level.values(){ }
```

Enum Fields / Methods :
constructor in Enum can only be private or package scoped.

```
public enum Level {
    HIGH(3), MEDIUM(2), LOW(1);
    private level int levelCode;
    private Level(int levelCode) {
        this.levelCode = levelCode;
    }

     public int getLevelCode() { return this.levelCode;}
}
```

Level level = Level.HIGH; level.getLevelCode() will be 3.

Java enum class can have abstract methods too.
All instances of enum class must implement that abstract method.
Java enum class can implement interfaces.
Java enum can not extend any class since it already extends java.lang.Enum class.

EnumSet and EnumMap to hold enums.
```
EnumSet<Level> enumSet = EnumSet.of(Level.HIGH, Level.MEDIUM);
EnumMap<Level, String> enumMap = new EnumMap<Level, String>(Level.class);
```

LAMBDA

Java lamdba expression is a function which can be created without belonging to any class.

Typically used to implement simple event listeners / callbacks / java stream apis.

Java lambda expressions can be used where the type they are matched against is a single unimplemented method interface (functional interfaces)
Parameters and return type of single unimplemented method interface should match with lambda.

Lambdas and Anonymous classes are almost similar except that Anonymous classes can have state (member variables). A lambda can not have that fields which makes it stateless.
"this" inside anonymous class refers to the anonymous class itself where as "this" inside lambda refers to enclosing object. lambdas can not have its own instance variables.

(a1, a2) -> { return a1>a2;} . can also be written as (a1, a2) -> a1 > a2;

A java lambda expression is essentially an object.
```
public interface MyComparator {
  public boolean compare(int a1, int a2);
}

MyComparator comp = (a1, a2) -> { return a1>a2};
comp.compare(a1, a2);
```

Local variables can be captured if and only if they are final or effectively final (String val = "abc";)
Static variables from enclosing class are also accessible from lambda.

Method References as Lambda:

If lambda has single sentence like "System.out.println(var);" then this can be replaced by simply:
System.out::println