JAVA Lambda and Annotations

Lambda / Functional Interfaces

- Lambda Expression
  - Helps us to write our code in functional style
  - Clear and concise way to implement Single Abstract Method

- Functional Interface
  - An interface which has only one abstract method is called functional interface
  - Annotation define ⇒ @FunctionalInterface
- Syntax
  - (argument-list) -> {body}

Example

```
1.  () -> {
2.  //Body of no parameter lambda
3.  }
```

```
@FunctionalInterface
interface Drawable{
	public void draw();
}

public class Example{


	public static void main(String... a){
		int width = 10;

		Drawable d2 = ()->{
			System.out.print("Widht"+width);
		};
		d2.draw();

	}

}
```

ForEach...

Advantages:

- It makes the code more readable.
- It eliminates the possibility of programming errors.

```
public static void main(String args[]){
   //declaring an array
   int arr[]={12,13,14,44};
   //traversing the array with for-each loop
   for(int i:arr){
     System.out.println(i);
   }
 }
```

Annotations

- Java Annotations allow us to add metadata information into our source code, although they are not a part of the program itself
- Annotations were added to the java from JDK 5
- Annotation has no direct effect on the operation of the code they annotate (i.e. it does not affect the execution of the program).

Use of Annotations

- **Instructions to the compiler**
  - **Eg: @Override, @Deprecated, @SupressWarnings**
- **Compile-time instructors**
  - **Eg: generating xml file**
- **Runtime instructions**
  - **Eg: Java Reflections**

Built-in annotations

Java.lang.Override

Compiler enforces overriding superclass methods

Java.lang.Deprecated

Warns when @deprecated method, class or member is used

Must use javac “-deprecation” argument to javac or the new -Xlint:deprecated flag to see compiler warnings

Java.lang.SupressWarnings

Useful for many compiler warnings (especially with legacy collection code)

Key: Look for a string enclosed in “[ ]” in compiler warnings from javac (may need to use the command line).

@OVERRIDE EXAMPLE

```
class Base {
    public void yes(int i) { }
}

class Subclass extends Base {
    @Override
    public void yes(int i) { }

    @Override
    public void no(float x) { }  // Compiler error generated here!
}
```

@DEPRECATED EXAMPLE

```
@Deprecated
public class DeprecatedExample {
   private int x;

   @Deprecated
   public int value;

  @Deprecated
   public void   setValue(int aValue) { value = aValue; }
}

public class DeprecatedExampleUse {
   public static void main(String argv[])
   {
      DeprecatedExample obj = new DeprecatedExample();  // Compiler WARNINGs generated here
      obj.value = 10;    // Compiler WARNING generated here
   }}
```

@SUPPRESSWARNING EXAMPLE

```
public class SuppressExample {
      List wordList = new ArrayList();    // no typing information on the List

      private void
      generateWarning()
      {
          wordList.add("foo");  // Warning generated here.
      }
}
```

```
javac -Xlint:unchecked SuppressExample.java
SuppressExample.java:13: warning: [unchecked] unchecked call to add(E) as a member of the raw type
java.util.List
          wordList.add("foo");  // Warning generated here.
                   ^
1 warning
```

Get rid of warning by adding this line above the method delcaration:

@SuppressWarning(“unchecked”)

Custom Annotations

- Annotations are created by using **@interface**, followed by annotation name as shown in the below example.
- An annotation can **have elements** as well. They look like methods. For example in the below code, we have four elements. We should not provide implementation for these elements.
- All annotations extends java.lang.annotation.Annotation interface
- Annotations cannot include any extends clause

Example

```java
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation{
    int studentAge() default 18;
    String studentName();
    String stuAddress();
    String stuStream() default "CSE";
}
```

```java
@interface MyCustomAnnotation {
    int      count();
    String[] books();
}
```

@Target

ElementType.METHOD

ElementType.PACKAGE

ElementType.PARAMETER

ElementType.TYPE

ElementType.ANNOTATION_TYPE

ElementType.CONSTRUCTOR

ElementType.LOCAL_VARIABLE

ElementType.FIELD

@Retention

RetentionPolicy.RUNTIME: The annotation should be available at runtime, for inspection via java reflection.

RetentionPolicy.CLASS: The annotation would be in the .class file but it would not be available at runtime.

RetentionPolicy.SOURCE: The annotation would be available in the source code of the program, it would neither be in the .class file nor be available at the runtime.

Example

```
import java.lang.annotation.Documented;
import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
@Documented
@Target(ElementType.METHOD) //TYPE, METHOD, CONSTRUCTOR, FIELD
@Inherited
@Retention(RetentionPolicy.RUNTIME)//SOURCE, CLASS and RUNTIME
public @interface MethodInfo{
        String author() default "sakthi";
        String date();
        int revision() default 1;
        String comments();
}
```

```
public static void main(String[] args) {
try {
for (Method method : AnnotationParsing.class.getClassLoader()
.loadClass(("AnnotationExample")).getMethods()) {
// checks if MethodInfo annotation is present for the method
if (method.isAnnotationPresent(MethodInfo.class)) {
try {
// iterates all the annotations available in the method
for (Annotation anno : method.getDeclaredAnnotations()) {
System.out.println("Annotation in Method '" + method + "' : " + anno);
}
MethodInfo methodAnno = method.getAnnotation(MethodInfo.class);
if (methodAnno.revision() == 1) {
System.out.println("Method with revision no 1 = " + method);
}

} catch (Throwable ex) {
ex.printStackTrace();
}}}
} catch (SecurityException | ClassNotFoundException e) {
e.printStackTrace();
}
}
```