Annotations are metadata that provide additional information about the code, but donâ€™t change how the code runs.

They act like labels or tags attached to your code that:
1.Help the compiler
2.Help tools (like frameworks or test runners)
3.Sometimes are used at runtime using Reflection

Common annotations:

Annotation	Purpose
@Override	Tells the compiler you're overriding a method
@Test	Used by JUnit to mark test methods
@SuppressWarnings	Tells compiler to ignore certain warnings

Why Do We Need Annotations?

Use Case	Description
âœ” Code-level documentation	Annotations make code self-descriptive
âœ” Compiler instructions	E.g., @Override causes a compile error if you donâ€™t override properly
âœ” Runtime behavior	Frameworks like Spring or JUnit use annotations to drive behavior
âœ” Reduce boilerplate code	Instead of XML or long code, annotations can configure behavior concisely

What Makes Up a Custom Annotation?

Element	Meaning
@Retention(...)	When the annotation is available (SOURCE, CLASS, RUNTIME)
@Target(...)	Where it can be applied (METHOD, FIELD, TYPE)
Method-like syntax	To define parameters for the annotation

What is RetentionPolicy?

@Retention(...) tells the compiler how long your annotation should be retained â€” i.e., where it should be available and visible.
You choose one of these:
@Retention(RetentionPolicy.SOURCE)
@Retention(RetentionPolicy.CLASS)
@Retention(RetentionPolicy.RUNTIME)

1. RetentionPolicy.SOURCE (Compile-time only)

The annotation is only present in source code
It gets discarded during compilation
Used mostly by tools like @Override, @SuppressWarnings

Example:
```
@Override
public String toString() { return "Hi"; }
```

âœ… Checked at compile-time

âŒ Not available in .class file or at runtime

2. RetentionPolicy.CLASS (Stored in .class, but not accessible at runtime)

Annotation is compiled and stored in the .class file
But itâ€™s not available to your running program
Useful for frameworks that work with bytecode (e.g., some compilers, annotation processors)
âš ï¸ This is the default if you donâ€™t specify @Retention

3. RetentionPolicy.RUNTIME (Available at runtime via reflection)

Annotation is present in source, compiled to .class, and available at runtime
âœ… This is used when your program/framework (like Spring, JUnit, Hibernate) needs to read annotations dynamically

Example:

```
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    String value();
}
```

Now you can read it via:

```
Method m = obj.getClass().getMethod("someMethod");
MyAnnotation ann = m.getAnnotation(MyAnnotation.class);
```

Comparison Table

Retention	Present in .java	Present in .class	Accessible at runtime
SOURCE	âœ… Yes	âŒ No	âŒ No
CLASS (default)	âœ… Yes	âœ… Yes	âŒ No
RUNTIME	âœ… Yes	âœ… Yes	âœ… Yes

When to Use Each?

Use Case	Retention Policy
Compiler checks like @Override	SOURCE
Bytecode tools or build-time processing	CLASS
Reflection or frameworks (JUnit, Spring)	RUNTIME

Creating custom annotation and using it:

We'll create a custom annotation @Info that stores a name and version, and then fetch those values using Reflection at runtime.

Step 1: Define the Annotation

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

@Retention(RetentionPolicy.RUNTIME) // Make it available at runtime
@Target(ElementType.METHOD)         // Can be used on methods
public @interface Info {
```

```
    String name();
    double version() default 1.0;
}
```

Step 2: Use the Annotation in a Class

```
public class MyService {

    @Info(name = "FetchData", version = 1.5)
    public void fetchData() {
        System.out.println("Fetching data...");
    }

    @Info(name = "SaveData")
    public void saveData() {
        System.out.println("Saving data...");
    }
}
```

Step 3: Read Annotation Values via Reflection

```
import java.lang.reflect.Method;

public class AnnotationReader {
    public static void main(String[] args) throws Exception {
        MyService service = new MyService();
        Class<?> clazz = service.getClass();

        for (Method method : clazz.getDeclaredMethods()) {
            if (method.isAnnotationPresent(Info.class)) {
                Info info = method.getAnnotation(Info.class);
                System.out.println("Method: " + method.getName());
                System.out.println("  Name: " + info.name());
                System.out.println("  Version: " + info.version());
            }
        }
    }
}
```

Output:

```
Method: fetchData
  Name: FetchData
  Version: 1.5

Method: saveData
  Name: SaveData
  Version: 1.0
```