Pattern Matching for Instanceof

An instanceof conditional check is generally followed by a typecasting.Java 14, gets rid of this verbosity by making conditional extraction a lot more concise.
Code:

```
public class InstanceOfDemo {
 public static void main(String[] args) {
 Object obj = (Object) "abc";

 if (obj instanceof String) {
 String abc = (String) obj;
 System.out.println("Old way " + abc);
 }

 if (obj instanceof String abc) {
 //String abc = (String)obj;
 System.out.println("Is string empty " + abc.isEmpty());
 System.out.println("New way " + abc);
 }
 }
}
```

Output:

```
Old way abc

Is string empty false

New way abc
```

TextBlockPreview

Text Blocks were introduced as a preview feature in Java 13 with the goal to allow easy creation of multiline string literals. It's useful in easily creating HTML and JSON or SQL query strings.

- In Java 14, Text Blocks are still in preview with some new additions. We can now use Backslash for displaying nice-looking multiline string blocks.
- \s is used to consider trailing spaces which are by default ignored by the compiler. It preserves all the spaces present before it.

Code:

```
public class TextBlockPreview {
 public static void main(String[] args) {
 String text = """
 Did you know \
 Java 14
has the most features among\
 all non-LTS versions so far\
 """;

 String text2 = """
 line1
 line2 \s
 line3
 """;

 String text3 = "line1\nline2 \nline3\n";

 System.out.println("Text " + text);
 System.out.println("Text2 is equal to Text3 " + text2.equals(text3));
 }
}
```

Output:

```
Text Did you know Java 14

has the most features amongall non-LTS versions so far

Text2 is equal to Text3 false
```

Sealed Classes

It offer developers of a Java class or interface the possibility to restrict which other classes and interfaces can extend or implement them.

A sealed class structure is defined as follows:

1.) The sealed keyword marks a sealed class.

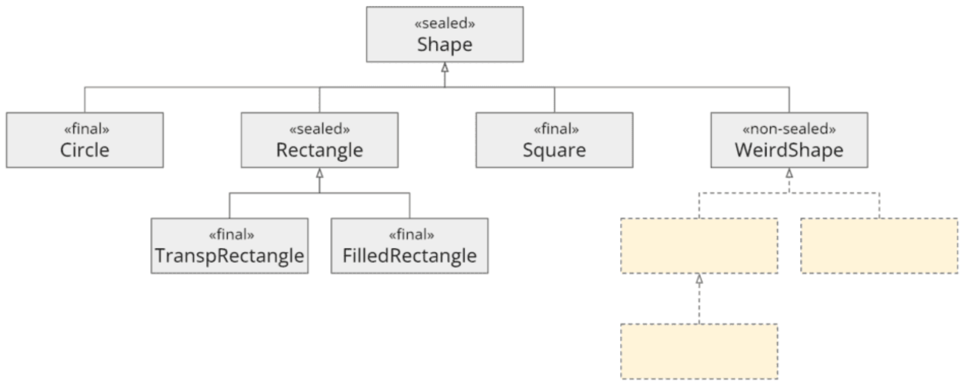
2.) After the keyword permits, you list the allowed subclasses.

3.) A subclass of a sealed class must be either `sealed`, `final`, or `non-sealed`. In the first case, you must again define the allowed subclasses with `permits`. The last case means that the subclass is again open to inheritance – just like any regular class.

Code:

```
public sealed class Shape permits Circle, Square, Rectangle, WeirdShape { }
public final class Circle extends Shape { }
public final class Square extends Shape { }
public sealed class Rectangle extends Shape
 permits TransparentRectangle, FilledRectangle { }
public final class TransparentRectangle extends Rectangle { }
public final class FilledRectangle extends Rectangle { }

public non-sealed class WeirdShape extends Shape { }
```



Records

A record is a data class that stores pure data. The idea behind introducing records is to quickly create simple and concise classes devoid of boilerplate code.

It is 2nd preview of records, the first preview was introduced in java 14 version.

Code:

```
public class RecordDemo {
 record UserRecord(int userId, String userName){

 }

 public static void main(String[] args) {
 UserRecord ur = new UserRecord(1342,"ABC");
 System.out.println(ur.userId());
 System.out.println(ur.userName());
 }
}
```

Output:

```
1342

ABC
```

Edwards-Curve Digital Signature Algorithm(EdDSA)

EdDSA is a modern signature method that is faster than previous signature methods, such as DSA and ECDSA while maintaining the same security strength. EdDSA is supported by many crypto libraries such as OpenSSL and BoringSSL. Many users already use EdDSA certificates.

Code:

```
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.Base64;
```

```
public class EDSAAlgo {
 public static void main(String[] args) throws NoSuchAlgorithmException,
InvalidKeyException, SignatureException {
 String message = "Happy Coding!";

 KeyPairGenerator kpg = KeyPairGenerator.getInstance("Ed25519");
 KeyPair kp = kpg.generateKeyPair();

 Signature sig = Signature.getInstance("Ed25519");
 sig.initSign(kp.getPrivate());
 sig.update(message.getBytes(StandardCharsets.UTF_8));
 byte[] signature = sig.sign();

 System.out.println("signature = " +
Base64.getEncoder().encodeToString(signature));
 }
}
```

Output:

```
 signature =
1W22woEw6sqVK94gewbs4ETJnICP9a8+76Lut/TP0ZkXR8lK8iqYrgbY8VVEJNRkOq/ZqSeJ
yxeOhTFbiamVCQ==
```