Java15/Examples

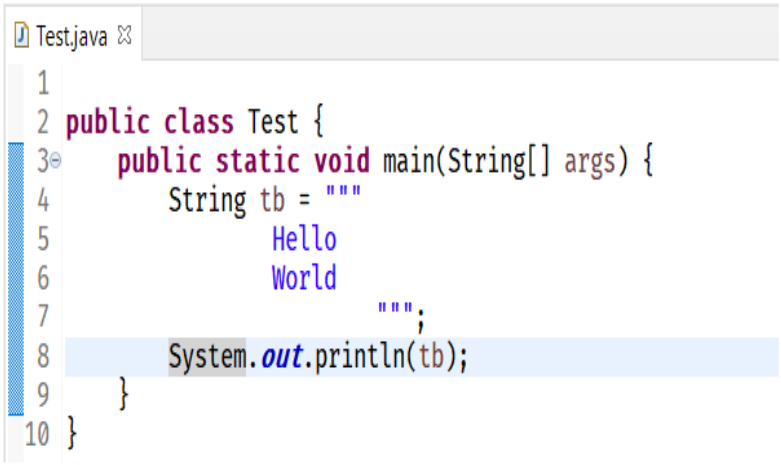
This is an informal page listing examples of features that are implemented by the Java 15 Support, which can be installed from the Marketplace (https://marketplace.eclipse.org/content/java-15-support-eclipse-2020-09-417). You are welcome to try out these examples. If you find bugs, please file a bug after checking for a duplicate entry here (https://bit.ly/2Ri7oHX)

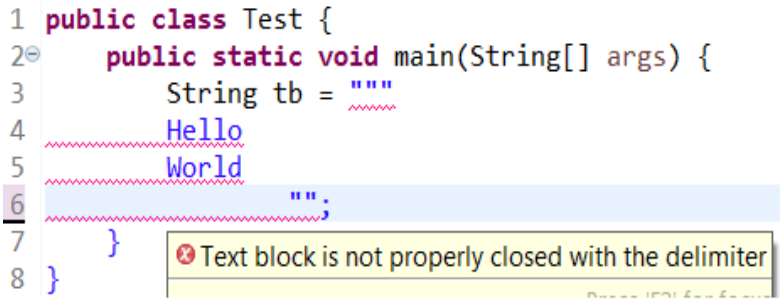
Watch out for additional examples being added soon.

NOTE:

- TextBlock is **standard features** in Java 15.
- Records is also another preview feature in Java 15. They are not enabled by default and can by enabled using **--enable-preview**.
- Instanceof Pattern Matching is also another preview feature in Java 15. They are not enabled by default and can by enabled using **--enable-preview**.
- In Eclipse, **--enable-preview** can be enabled from the Preferences. It is implicitly added while launching a java program if the feature has been enabled for the project/workspace.

	Feature / Steps	**Expected Result**
Standard Java/JRE setup with Eclipse	More details can be found on dedicated Java/JRE setup (https://wiki.eclipse.org/Java/Setup) page.	Setting up Java/JRE in Eclipse.
Overview of eclipse.ini	More details can be found on dedicated eclipse.ini (https://wiki.eclipse.org/Eclipse.ini) page.	Specifying the JVM in eclipse.ini
Standard Feature: Text Blocks.		

Text Block Example	Compile and run the following code:
	`public class Test {`
	`  public static void main(String[] args) {`
	`  String tb = """`
	`  Hello`
	`  World`
	`  """;`
	`  System.out.println(tb);`
	`  }`
	`}`
	 
	(/File:Textblock.png)
		Code compiles and prints both "Hello" "World" as it is - notice that "World" is printed in the next line.

Text Block Compilation Error Example	Use the following code:
	`public class Test {`
	`  public static void main(String[] args) {`
	`  String tb = """`
	`  Hello`
	`  World`
	`  "";`
	`  }`
	`}`
	 
	(/File:Textblock.error.png)
		Compilation error - text block not closed properly

Preview Feature: Records		

Postive compilation1 (Record Example)	Compile and run the following code:
	`@SuppressWarnings("preview")`
	`record Point(int x, int y) {`
	`}`
	`public class X1 {`
	`  public static void main(String[] args) {`
	`  Point p = new Point(100, 200);`
	`  System.out.println(p.x());`
	`  }`
	`}`
		Code compiles and prints 100.

Positive compilation2 (Nested Record Example)	Compile and run the following code:
	`class X2 {`
	`  public static void main(String[] args) {`
	`  System.out.println(0);`
	`  }`
	`  @SuppressWarnings("preview")`
	`  record Point(int x, int y) {`
	`  }`
	`}`
		Code compiles and prints 0.

Positive compilation3 (Record Example)	Compile and run the following code:
	`class X3 {`
	`  public static void main(String[] args) {`
	`  System.out.println(0);`
	`  }`
	`}`
	`@SuppressWarnings("preview")`
	`final record Point(int x, int y) {`
	`}`
		Code compiles and prints 0. Though a record declaration is implicitly final, it is permitted for the declaration of a record type to redundantly specify the final modifier

Positive compilation4	Compile and run the following code:
	`@SuppressWarnings("preview")`
	`record R() {`
	`}`
	`class X4 {`
	`  public static void main(String[] args) {`
	`  System.out.println(new R().hashCode());`
	`  }`
	`}`
		Code compiles and prints 0.

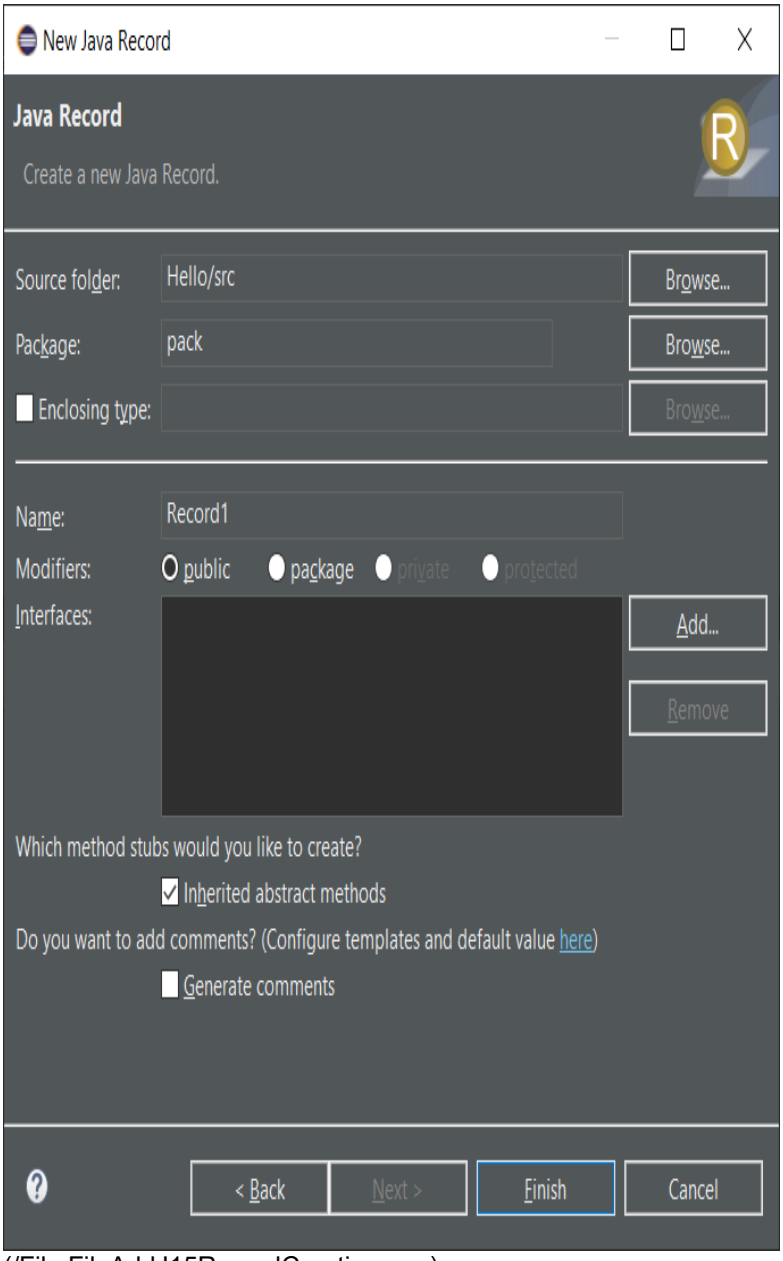
Positive compilation5	Compile and run the following code:
	`import java.lang.annotation.Target;`
	`import java.lang.annotation.ElementType;`
	`@Target({ ElementType.PARAMETER })`
	`@interface MyAnnot {`
	`}`
	`@SuppressWarnings("preview")`
	`record R(@MyAnnot()int i, int j) {`
	`}`
	`class X5 {`
	`  public static void main(String[] args) {`
	`  System.out.println(new R(100, 200).hashCode() != 0);`
	`  }`
	`}`
		Code compiles and prints true.

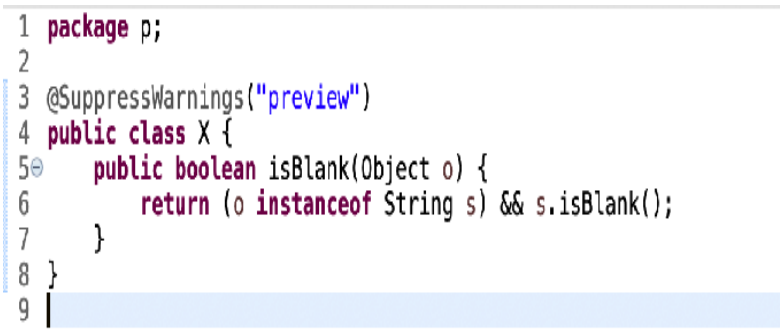
Positive compilation6	Compile and run the following code:
	`class X6 {`
	`  @SuppressWarnings("preview")`
	`  public static void main(String[] args) {`
	`  record R(int x,int y){}`
	`  R r = new R(100, 200);`
	`  System.out.println(r.x());`
	`  }`
	`}`
		Code compiles and prints 100.

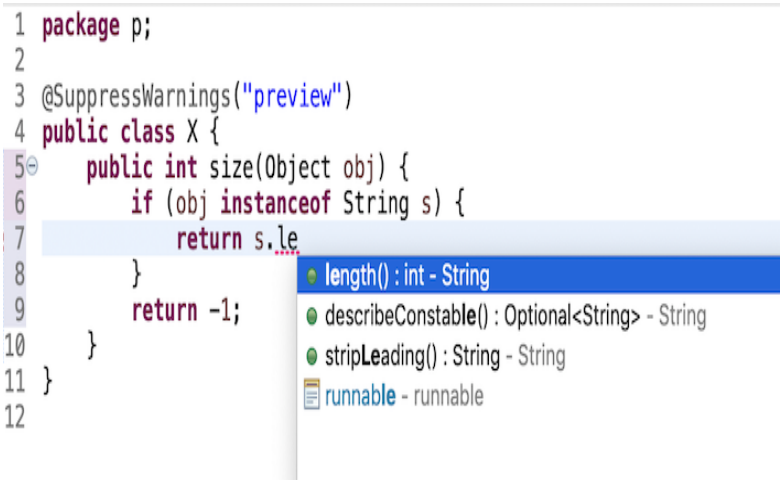
Negative compilation1 (Record Example)	Compile and run the following code:
	`@SuppressWarnings("preview")`
	`abstract record Point(int x, int y){`
	`}`
	`class X7 {`
	`  public static void main(String[] args){`
	`  System.out.println(0);`
	`  }`
	`}`
		Code fails to compile with error "Illegal modifier for the record Point; only public, final and strictfp are permitted"

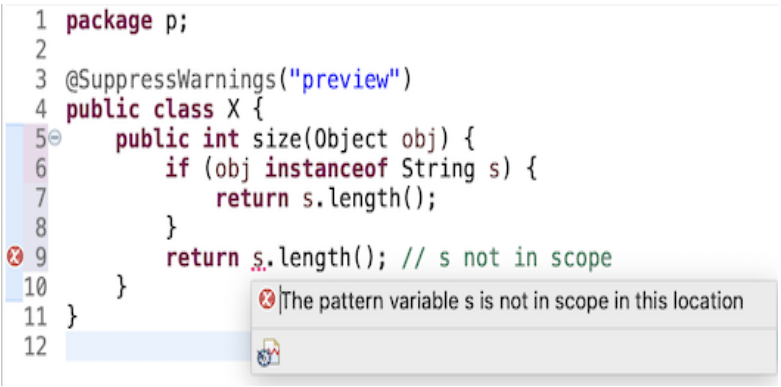
Negative compilation2 (Record Example)	Compile and run the following code:
	`@SuppressWarnings("preview")`
	`record Point1(int myInt, char myChar) implements I {`
	`public Point1 {`
	`this.myInt = myInt;`
	`this.myChar = myChar;`
	`}`
	`}`
	`public class X8 {`
	`public static void main(String[] args) {`
	`System.out.println(0);`
	`}`
	`}`
	`interface I {`
	`}`
		Code fails to compile with error "The canonical constructor Point1 of a record declaration must be declared public."

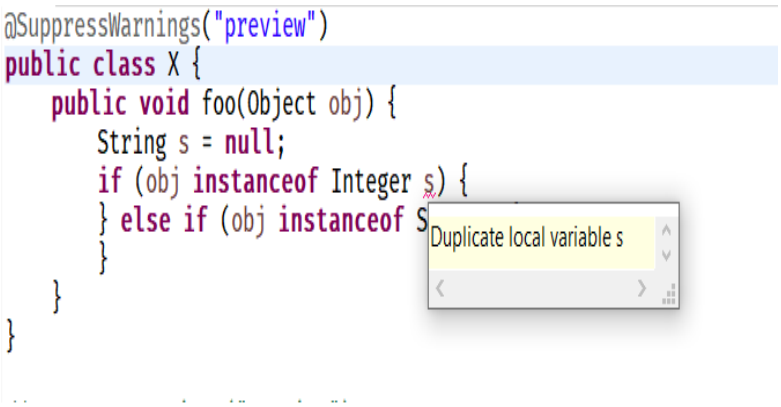
Negative compilation3 (Record Example)	Compile and run the following code:
	`class record {`
	`public static void main(String[] args) {`
	`System.out.println(0);`
	`}`
	`}`
		Code fails to compile with error "Record is a restricted identifier and hence not a valid type name"

Record Creation Wizard	Right Click on the Project -> New -> Record **or** Right Click on the Project -> New -> Other and search for Record **or** Right Click on the Project -> New -> Other -> Java -> Record  (/File:FileAddJ15RecordCreation.png) [note: In older workspaces Record option may not appear diectly under New menu in java perspective. To resolve this, use a new workspace or relaunch eclipse with -clearPersistedState for the same workspace]	Record is created
Preview Feature: Instanceof Pattern Matching		

Postive compilation1 (Instanceof Pattern Matching Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`public class X {`
	`  public boolean isBlank(Object o) {`
	`  return (o instanceof String s) && s.isBlank();`
	`  }`
	`}`
	 
	(/File:Pattern-match1.png)
		The pattern variable 's' is in current scope

Postive compilation2 (Instanceof Pattern Matching Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`public class X {`
	`  public int size(Object obj) {`
	`  if (obj instanceof String s) {`
	`  return s.le`
	`  }`
	`  return -1;`
	`  }`
	`}`
	 
	(/File:Pattern-match2.png)
		The pattern variable 's' is in current scope inside 'then' statement and completion proposes applicable methods on String, the pattern matched type.

Negative compilation1 (Instanceof Pattern Matching Example)	Use the following code:
	`package p;`
	
	`@SuppressWarnings("preview")`
	`public class X {`
	`  public int size(Object obj) {`
	`  if (obj instanceof String s) {`
	`  return s.length();`
	`  }`
	`  return s.length(); // s not in scope`
	`  }`
	`}`
	
	 
	
	(/File:Pattern-match3.png)
		The pattern variable 's' is rejected by the compiler when not in scope outside the 'then' statement.

Negative compilation2 (Instanceof Pattern Matching Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`public class X {`
	`  public void foo(Object obj) {`
	`  String s = null;`
	`  if (obj instanceof Integer s) {`
	`  } else if (obj instanceof String) {`
	`  }`
	`  }`
	`}`
	
	 
	
	(/File:Pattern-match4.png)
		The pattern variable 's' is rejected by the compiler as "Duplicate local variable s".

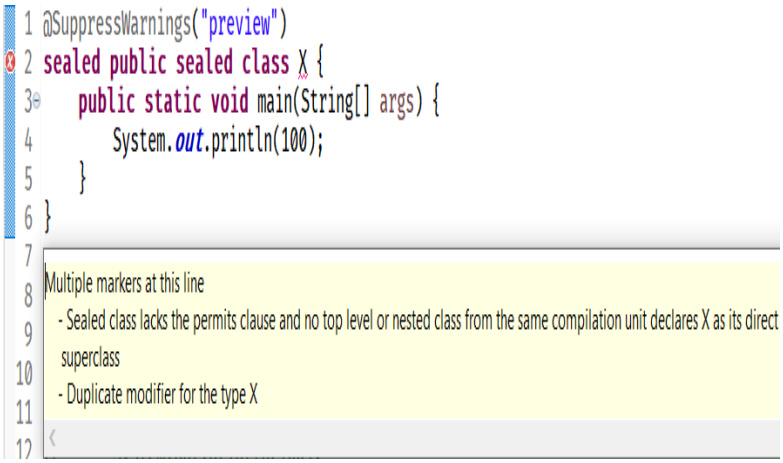
Preview Feature: Sealed Classes		

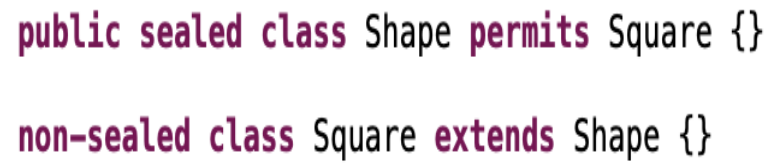
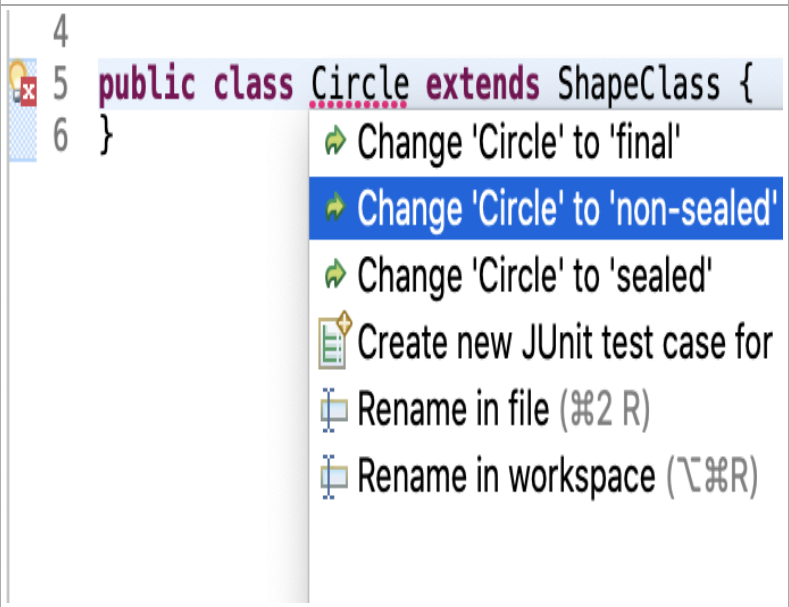
Postive compilation1 (Sealed Class Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`sealed class Y permits X {`
	`}`
	
	`@SuppressWarnings("preview")`
	`non-sealed class X extends Y {`
	`public static void main(String[] args) {`
	`System.out.println(0);`
	`}`
	`}`
	
	(/File:Sealed-class1.png)
		Code compiles and prints 0.

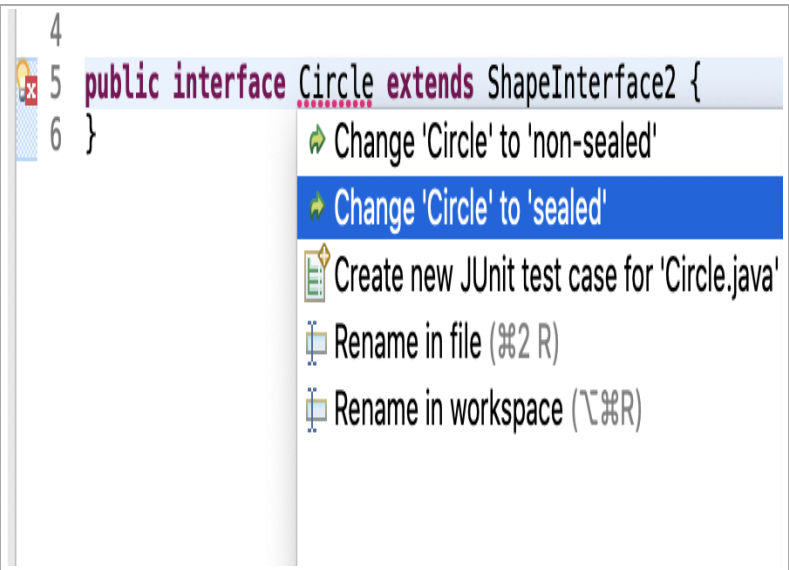
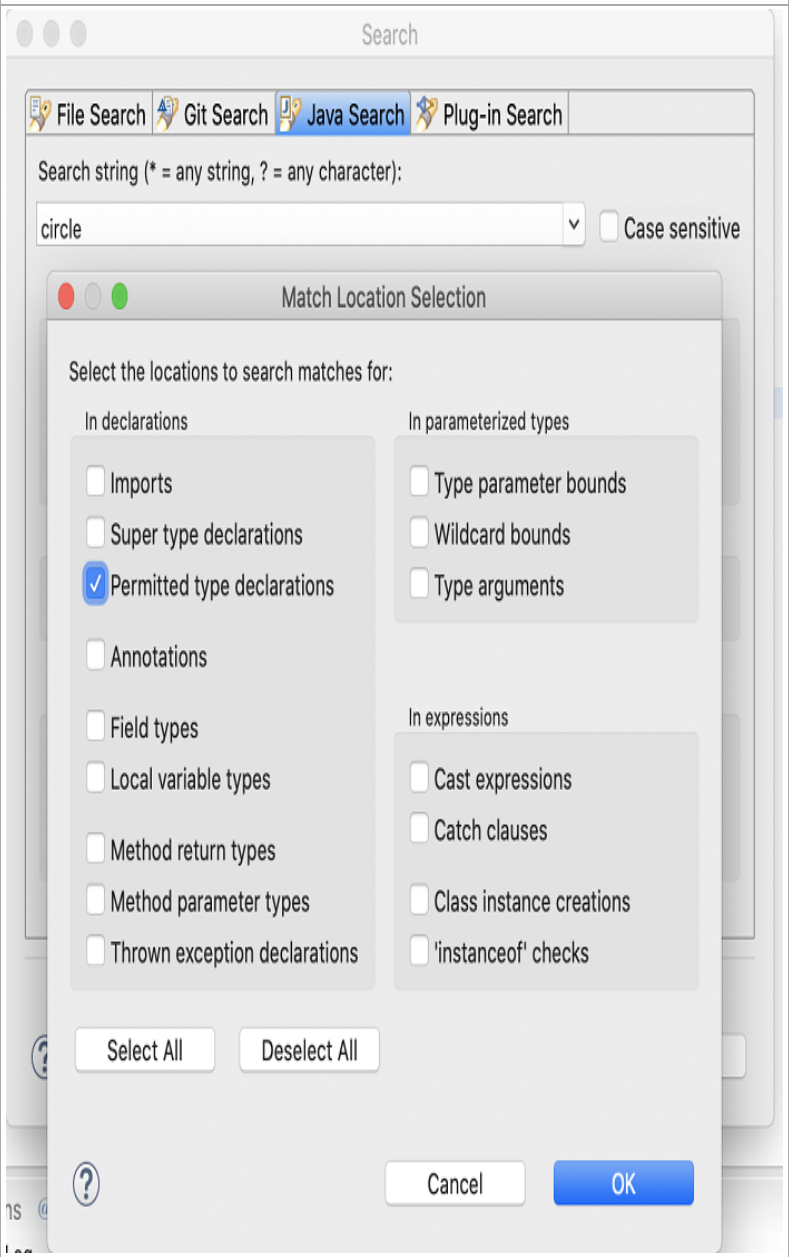
Postive compilation2 (Sealed Class Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`sealed interface I extends SI {`
	`}`
	
	`@SuppressWarnings("preview")`
	`non-sealed class X implements SI {`
	`public static void main(String[] args) {`
	`System.out.println(0);`
	`}`
	`}`
	
	`@SuppressWarnings("preview")`
	`sealed interface SI permits X,I {`
	`}`
	
	`@SuppressWarnings("preview")`
	`non-sealed interface I2 extends I {`
	`}`
	
	`@SuppressWarnings("preview")`
	`sealed interface I extends SI {`
	`}`
	
	`@SuppressWarnings("preview")`
	`non-sealed class X implements SI {`
	`public static void main(String[] args) {`
	`System.out.println(0);`
	`}`
	`}`
	
	`@SuppressWarnings("preview")`
	`sealed interface SI permits X,I {`
	`}`
	
	`@SuppressWarnings("preview")`
	`non-sealed interface I2 extends I {`
	`}`
	(/File:Sealed-class2.png)
		Code compiles and prints 0.

Postive compilation3 (Sealed Class Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`sealed class X permits Y {`
	`  public static void main(String[] args) {`
	`  System.out.println(100);`
	`  }`
	`}`
	`@SuppressWarnings("preview")`
	`non-sealed class Y extends X {`
	`}`
	
	(/File:Sealed-class3.png)
		Code compiles and prints 100.

Postive compilation4 (Sealed Class Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`sealed public class X<T> {`
	`  public static void main(String[] args) {`
	`  System.out.println(100);`
	`  }`
	`}`
	`@SuppressWarnings({ "preview", "rawtypes" })`
	`non-sealed class Y extends X {`
	`}`
	
	(/File:Sealed-class4.png)
		Code compiles and prints 100.

Negative compilation1 (Sealed Class Example)	Use the following code:
	`@SuppressWarnings("preview")`
	`sealed public sealed class X {`
	`public static void main(String[] args) {`
	`System.out.println(100);`
	`}`
	`}`
	 
	
	(/File:Sealed-class5.png)
		Code fails to compile with error "Multiple markers at this line
		- Sealed class lacks the permits clause and no top level or nested class from the same compilation unit declares X as its direct superclass
		- Duplicate modifier for the type X"

Syntax coloring for “sealed”, “non-sealed”, “permits”	 (/File:Syntax-coloring.png)	New restricted keywords are colored
Quick fixes support to add sealed/final/non-sealed modifier on a permitted class declaration.	 (/File:Quick-fixes-3.png)	Three quick fix modifier options: - sealed - non-sealed - final

Quick fixes support to add sealed/non-sealed modifier on a permitted interface declaration.	 (/File:Quick-fixes-2.png)	Two quick fix modifier options: - sealed - non-sealed
"Match locations" dialog supports "Permitted type declarations"	 (/File:Permitted-Type-Declarations.png)	"Search > Java Search > Match locations" dialog supports "Permitted type declarations" check-box option.