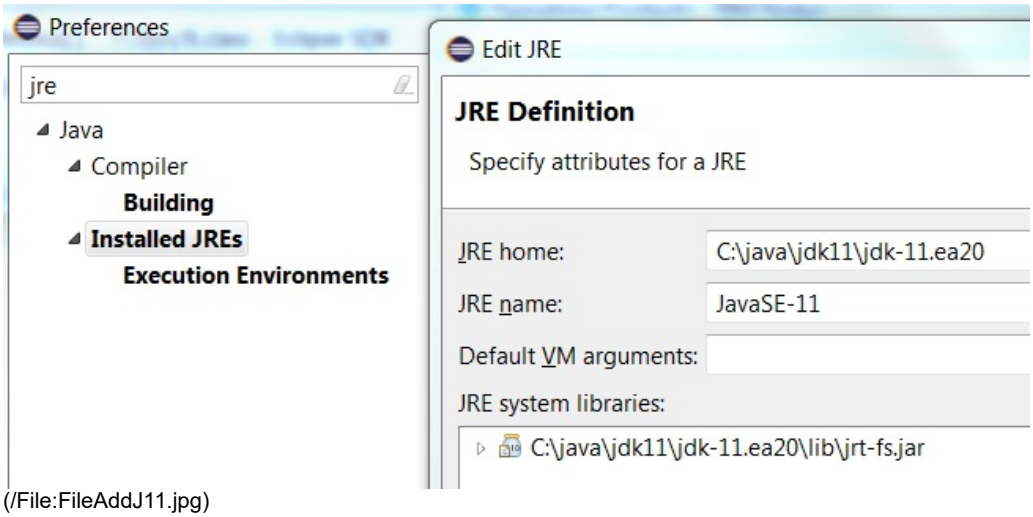
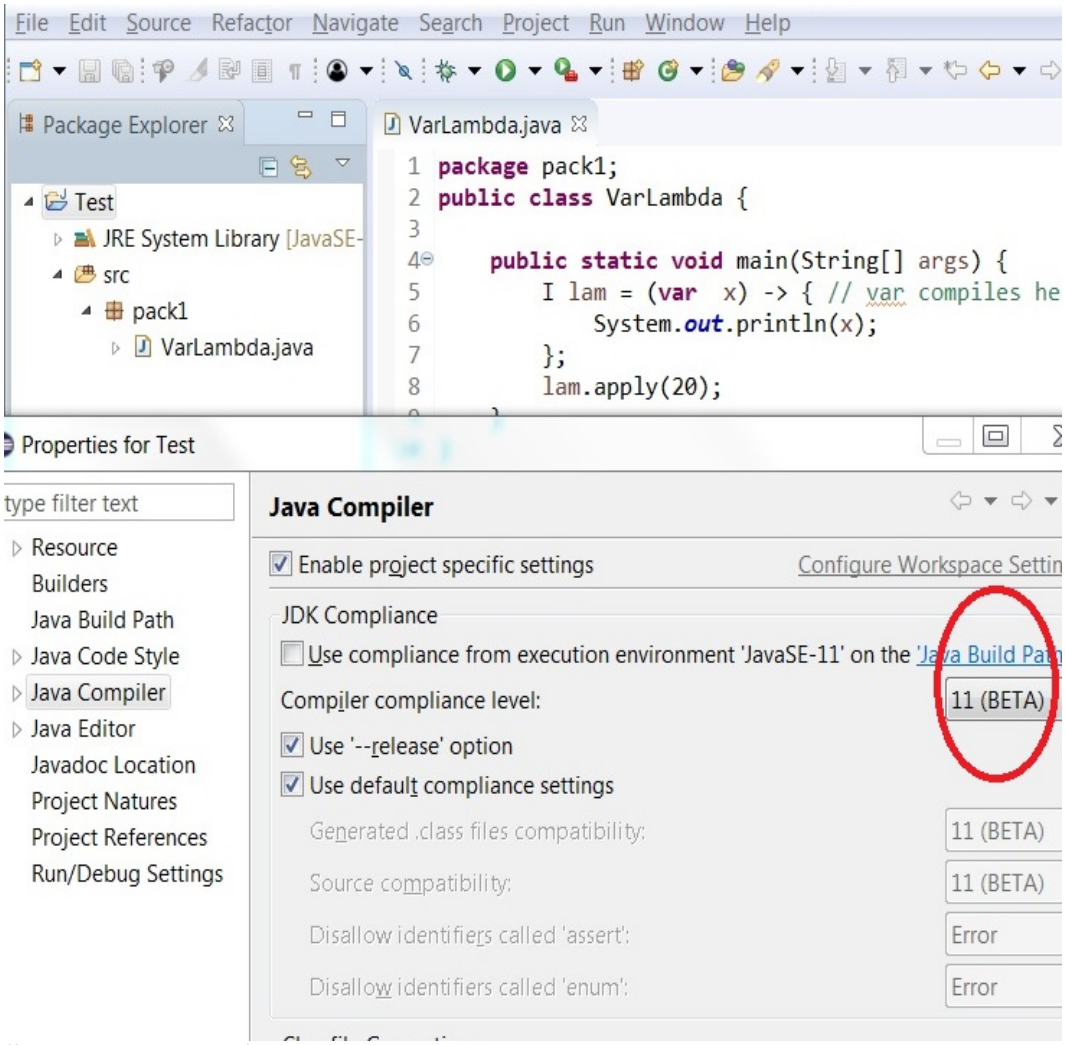
Java11/Examples

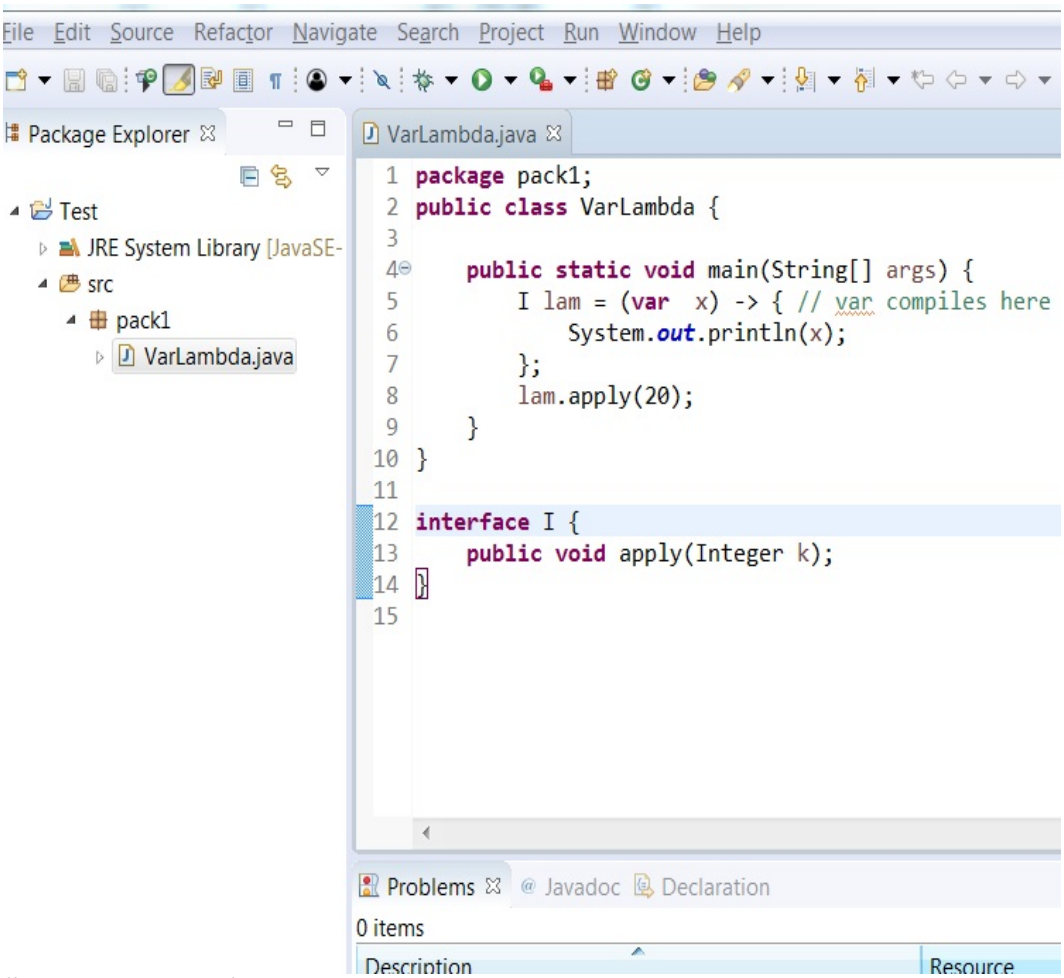
This is an informal page listing examples of features that are implemented by the Java 11 Support. You are welcome to try out these examples. If you find bugs, please file a bug after checking for a duplicate entry here (https://bugs.eclipse.org/bugs/buglist.cgi?cmdtype=dorem&remaction=run&namedcmd=J11.Open&sharer_id=152344).

	Feature / Steps	**Expected Result**
	The Pre-requisite: Java 11 JRE Support	
Add Java 11 JRE	Use Window -> Preferences-> Java -> Installed JREs -> Add...  (/File:FileAddJ11.jpg) [note: Eclipse -> Preferences in Mac / Window -> Preferences in Windows]	Java 11 JRE recognized as a valid JRE
Project JRE	In Package Explorer Use project's context menu and add Java 11 JRE	JRE specific (eg Object) gets resolved in the project.
Package Explorer	Go to Package Explorer and expand the Java 11 JRE	Modules (eg java.base etc) are listed in the package explorer view
	The First Step: Java 11 Compliance	
Set Project Compliance in Package Explorer	Context Menu of Project -> Properties -> Set project-specific, drop down to 11	11 is shown in the drop down list



(/File:J11.compliance.jpg)

Basic Necessity : Compilation and Error Reporting

Positive Compilation	Use the following code:
	`package pack1;`
	`public class VarLambda {`
	`public static void main(String[] args) {`
	`I lam = (var x) -> { // var compiles here`
	`System.out.println(x);`
	`};`
	`lam.apply(20);`
	`}`
	`}`
	`interface I {`
	`public void apply(Integer k);`
	`}`
	 
	
	(/File:Var11.compile.jpg)
		Code compiles

Compiler Error Case 1	`package pack1;`
	`public class VarLambdaErr {`
	`public static void main(String[] args) {`
	`I lam = (var x, z) -> { // should not mix with type-elided params`
	`System.out.println(x);`
	`};`
	`lam.apply(20);`
	`}`
	`}`
	`interface I {`
	`public void apply(Integer k, Integer z);`
	`}`
		Compiler errors are shown

Compiler Error Case 2	`package pack1;`
	`public class VarLambdaErr {`
	`public static void main(String[] args) {`
	`I lam = (var x, Integer z) -> { // should not mix with non-var params`
	`System.out.println(x);`
	`};`
	`lam.apply(20, 200);`
	`}`
	`}`
	`interface I {`
	`public void apply(Integer k, Integer z);`
	`}`
		Compiler errors shown

Hover and Navigation		

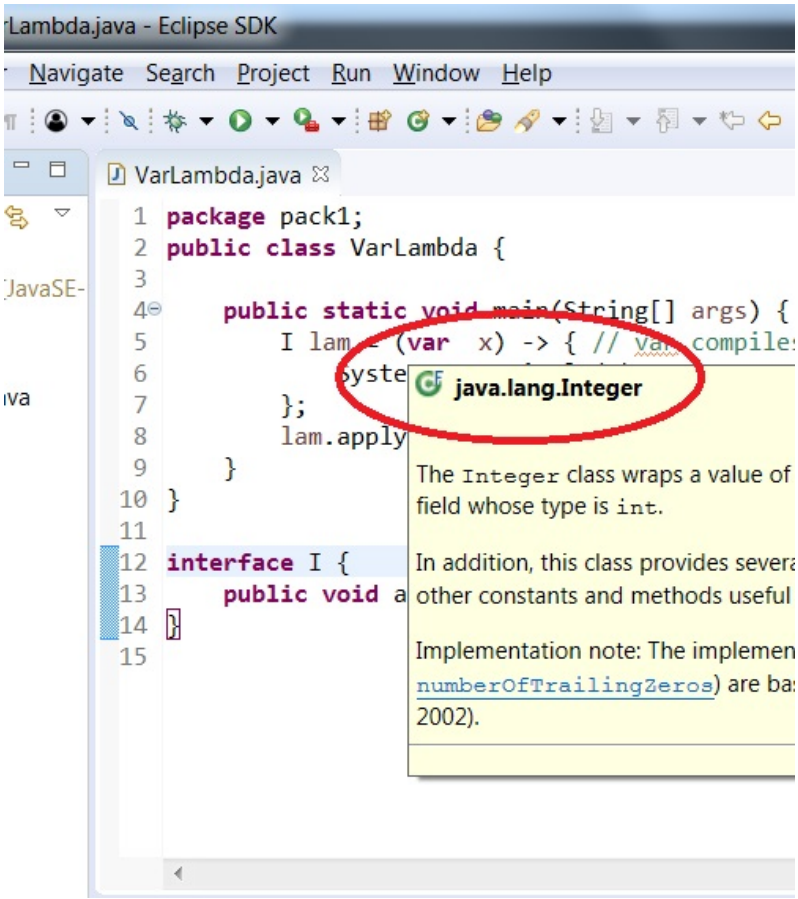
Hover

Use the following code:

```
package pack1;
public class VarLambda {

        public static void main(String[] args) {
                I lam = (var  x) -> { // hover over var
                        System.out.println(x);
                };
                lam.apply(20);
        }
}

interface I {
        public void apply(Integer k);
}
```



(/File:Var11.hover.jpg)

Hover and Navigation

Nestmates

Basic Nesting Principles

Description: Nest Based Access - A JVM Concept.
A top level class and all inner classes form a single unit for access, a "nest".JVM specification added two attributes in the classfile NestHost and NestMember. The top most class will have the NestMember attribute listing all the members while each of the inner classes will have a NestHost attribute listing the top level class. Using this, some of the synthetic bridge methods are elided transparent to the programmer. Note that this feature is relevant only for tools that process byte code and hence, in general, this feature would be "transparent" to a "normal" programmer. This feature is applicable for byte code processors, for eg our Disassembler has been enhanced to show these attributes. For further info please read JEP 181.

```
public class X {
   private class A {
    class B {}
  }
  private class Y extends A {
  }
}
```

In X.class (Disassembled) Nest Members:

```
#21 X$A,
#24 X$A$B,
#27 X$Y
```

In each of XA, XAB, XY the attribute: Nest Host: #22 X

This page was last modified 01:43, 13 March 2019 by Lakshmi Shanmugam (/index.php?title=User:Lshanmug.in.ibm.com&action=edit&redlink=1). Based on work by Manoj Palat (/index.php?title=User:Manpalat.in.ibm.com&action=edit&redlink=1).