

```

...
ActionListener clk = new ActionListener()
{
    public void click()
    {
        // --- implementation of the click event ---
        ...
        return;
    }
};

```

In the above example the class that implements the `ActionListener` is **anonymous**. The class is defined where it is instantiated.

The above code is harder to read than if the class explicitly defined, so why use it? If many implementations are needed for an interface and those classes are used only in one particular place, using **anonymous** class makes sense.

The following example uses anonymous class to implement an action listener.

```

import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
class MyApp implements Serializable
{
    BigObjectThatShouldNotBeSerializedWithAButton bigOne;
    Button aButton = new Button();
    MyApp()
    {
        aButton.addActionListener( new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                System.out.println("Hello There");
            }
        }
        );
    }
}

```

The following example does the same thing, but it names the class that implements the action listener.

```

import java.awt.*;
import java.awt.event.*;
import java.io.Serializable;
class MyApp implements Serializable
{
    BigObjectThatShouldNotBeSerializedWithAButton bigOne;
    Button aButton = new Button();
    // --- Nested class to implement the action listener ---
    class MyActionListener implements ActionListener
    {

```

```
public void actionPerformed(ActionEvent e)
{
    System.out.println("Hello There");
}

MyApp ()
{
    aButton.addActionListener( new MyActionListener() );
}
}
```

Using **anonymous** classes is especially preferable when you intend to use many different classes that each implement the same Interface.

Access Modifiers

Navigate **Language Fundamentals** topic:

Access modifiers

You surely would have noticed by now, the words **public**, **protected** and **private** at the beginning of class's method declarations used in this book. These keywords are called the **access modifiers** in the Java language syntax, and can be defined as...



.. keywords that help set the visibility and accessibility of a class, its member variables, and methods.



The following table shows what Access Modifiers are appropriate for classes, nested classes, member variables, and methods:

	Class	Nested class	Method, or Member variable	Interface	Interface method signature
public	visible from anywhere	same as its class	same as its class	visible from anywhere	visible from anywhere
protected	N/A	its class and its subclass	its class and its subclass, and from its package	N/A	N/A
package (default)	only from its package	only from its package	only from its package	only from its package	N/A, default is public
private	N/A	only from its class	only from its class	N/A	N/A



Points to ponder

Note that Interface method visibility is public by default. You do not need to specify the access modifier it will default to public. For clarity it is considered a good practice to put the public keyword. The same way all member variables defined in the Interface by default will become static final once inherited in a class.

If a class has public visibility, the class can be referenced by anywhere in the program. If a class has package visibility, the class can be referenced only in the package where the class is defined. If a class has private visibility,

(it can happen only if the class is defined nested in an other class) the class can be accessed only in the outer class.

If a variable is defined in a public class and it has public visibility, the variable can be reference anywhere in the application through the class it is defined in. If a variable has package visibility, the variable can be referenced only in the same package through the class it is defined in. If a variable has private visibility, the variable can be accessed only in the class it is defined in.

If a method is defined in a public class and it has public visibility, the method can be called anywhere in the application through the class it is defined in. If a method has package visibility, the method can be called only in the same package through the class it is defined in. If a method has private visibility, the method can be called only in the class it is defined in.

Methods

Navigate **Language Fundamentals** topic:

Method Definition

Method is an operation on a particular object. An object is an instance of a class. When we define a class we define its member variables and its methods. For each method we need to give a name, we need to define its input parameters and we need to define its return type. We also need to set its visibility(private, package, or public). If the method throws an Exception, that needs to be declared as well. The syntax of method definition is:

```
class MyClass
{
    ...
    public ReturnType methodName( ParamOneType param1, ParamTwoType
param2 ) throws ExceptionName
    {
        ReturnType retType;
        ...
        return retType;
    }
    ...
}
```

We can declare that the method does not return anything using the **void** java keyword. For example:

```
private void methodName( String param1, String param2 )
{
    ...
    return;
}
```

When the method returns nothing, the **return** keyword at the end of the method is optional. The **return** keyword can be used anywhere in the method, when the execution flow reach the **return** keyword, the method execution is stopped and the execution flow returns to the caller method.