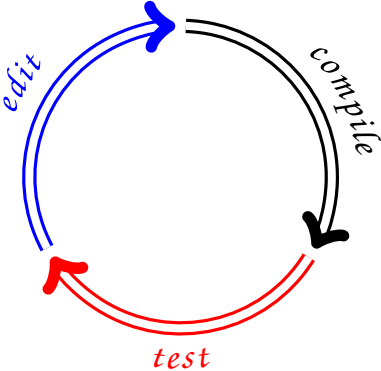


# Program Development



# Developing Java Programs – BlueJ

The image displays the BlueJ IDE interface. On the left, a code editor shows the implementation of the `Student` class, which inherits from `Person`. The code includes comments, author information, and a constructor that initializes a student with a name and ID.

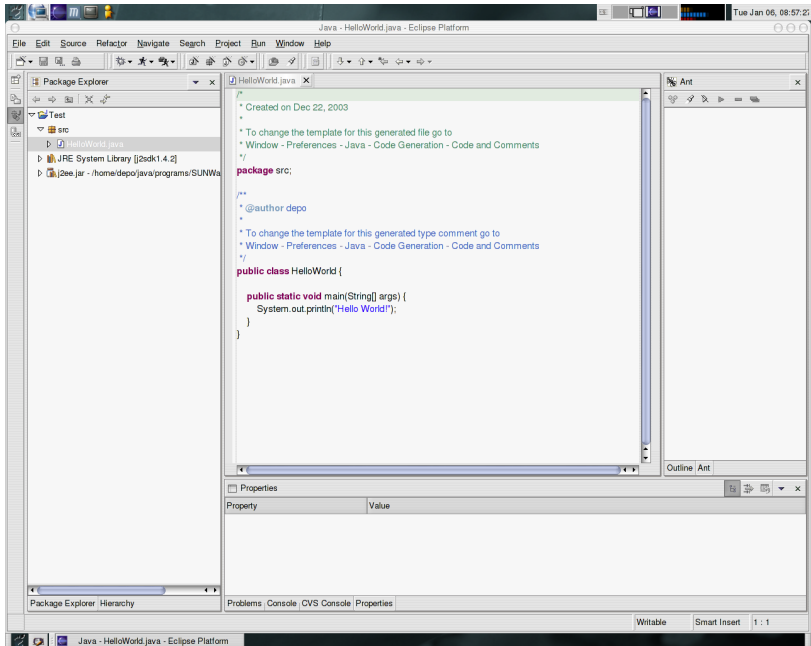
```
/**
 * A class representing students for a simple
 *
 * @author Michael Kolling
 * @version 1.0, January 1999
 */
class Student extends Person
{
    private String SID; // student ID number

    /**
     * Create a student with default settings
     */
    public Student()
    {
        super("(unknown name)", 0000);
        SID = "(unknown ID)";
    }

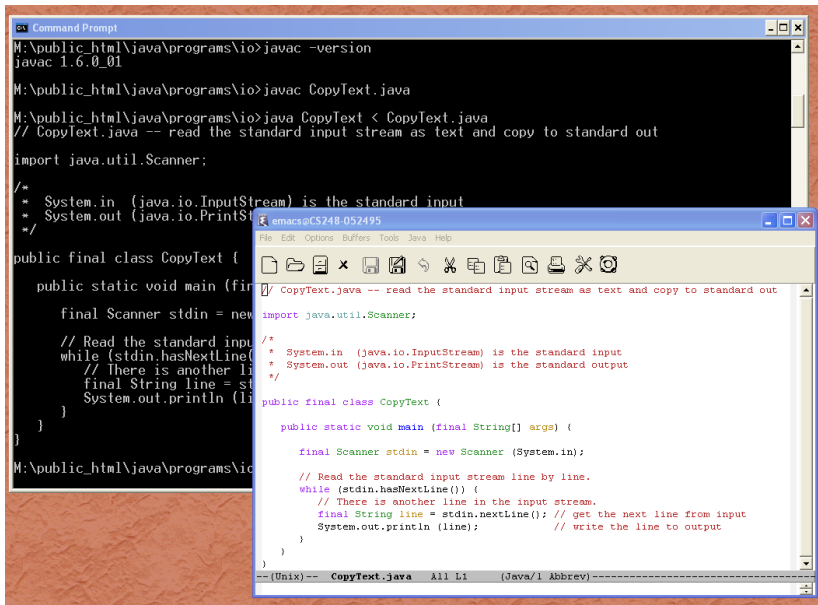
    /**
     * Create a student with given name, year
     */
}
```

On the right, the UML class diagram for the `people2` project is shown. It features an abstract class `Person` with two subclasses, `Staff` and `Student`. The `Person` class has a dashed association to an `Address` class. The `Database` class is also shown with a dashed association to `Person`. A context menu is open over the `Person` class, listing its methods: `String getRoom()`, `void setRoom(String)`, and `String toString()`. The menu also shows that `Person` is inherited from `Object` and `Person`. At the bottom of the IDE, there are buttons for `student_1: Student` and `staff_1: Staff`, and a `saved` button.

# Developing Java Programs – Eclipse



# Developing Java Programs – Emacs



```
M:\public_html\java\programs\io>javac -version
javac 1.6.0_01

M:\public_html\java\programs\io>javac CopyText.java

M:\public_html\java\programs\io>java CopyText < CopyText.java
// CopyText.java -- read the standard input stream as text and copy to standard out

import java.util.Scanner;

/*
 * System.in (java.io.InputStream) is the standard input
 * System.out (java.io.PrintStream) is the standard output
 */

public final class CopyText {

    public static void main (final String[] args) {

        final Scanner stdin = new Scanner (System.in);

        // Read the standard input stream line by line.
        while (stdin.hasNextLine()) {
            // There is another line in the input stream.
            final String line = stdin.nextLine(); // get the next line from input
            System.out.println (line); // write the line to output
        }
    }
}

M:\public_html\java\programs\io>
```

```
emacs@CS248-052495
File Edit Options Buffers Tools Java Help

// CopyText.java -- read the standard input stream as text and copy to standard out
import java.util.Scanner;

/*
 * System.in (java.io.InputStream) is the standard input
 * System.out (java.io.PrintStream) is the standard output
 */

public final class CopyText {

    public static void main (final String[] args) {

        final Scanner stdin = new Scanner (System.in);

        // Read the standard input stream line by line.
        while (stdin.hasNextLine()) {
            // There is another line in the input stream.
            final String line = stdin.nextLine(); // get the next line from input
            System.out.println (line); // write the line to output
        }
    }
}

--(Unix)-- CopyText.java All L1 (Java/1 Abbrev)--
```

- ▶ compile error
  - ▶ syntax error — example program
  - ▶ semantic error — example program
    - ▶ type error — example program

- ▶ Eclipse warnings

- ▶ style error — example program

Style errors are mistakes in the program source code that contravene policy or hamper the ability of programmers to read and understand the program even though the program can be translated by the compiler into an executable program.

- ▶ execution error or (fatal) runtime error — example program

Runtime errors are mistakes that manifest themselves during the execution of the program. These errors prevent the computer from completing the execution of the program.

- ▶ logic error — example program

Logic errors are mistakes in the behavior of the program even though the program can be translated into a running, executable program.

Java requires many suspicious behaviors to be flagged as errors (not just warnings).

According to the Java Language Specification:

“It is a compile-time error if a statement cannot be executed because it is unreachable.”

Java has optional warnings enabled by `javac -Xlint`

In Java 1.6 the complete list (obtained by `javac -X`):

`cast, deprecation, divzero, empty, unchecked, fallthrough, path, serial, finally, overrides`

The warnings `deprecation` and `unchecked` are checked in all cases (regardless of the command line options).

```
java -Xlint:all -Xlint:-serial
```

The screenshot shows the Eclipse IDE interface. The main editor displays the following Java code:

```
import java.io.BufferedReader;

public class DemoClass {
    public static void main(String args[]) {
        int variable1 = 1;
        int variable2 = 2;
        int variable3 = 3;
    }
}
```

A tooltip popup is visible over the line `int variable3 = 3;` with the message: "The value of the local variable variable3 is not used". It lists four quick fixes:

- Remove 'variable3' and all assignments
- Remove 'variable3', keep side-effect assignments
- Fix 2 problems of same category in file
- Add @SuppressWarnings('unused') to 'variable3'
- Add @SuppressWarnings('unused') to 'main()'

The Problems view at the bottom shows three warnings:

Description	Path	Location	Type	
The import java.io.BufferedReader is never used	DemoClass.java	/DemoProject/src	line 1	Java Problem
The value of the local variable variable1 is not used	DemoClass.java	/DemoProject/src	line 5	Java Problem
The value of the local variable variable3 is not used	DemoClass.java	/DemoProject/src	line 7	Java Problem

The status bar at the bottom indicates: "The value of the local ...e variable3 is not used | Writable | Smart Insert | 7 : 21".

Eclipse warns about semantic problems not required by the Java language specification

If you make a mistake and write a program that goes into an endless loop, and the computer runs out time or space resources and terminates your program prematurely, is this a runtime or a logic error? Either, both, what difference does it make?

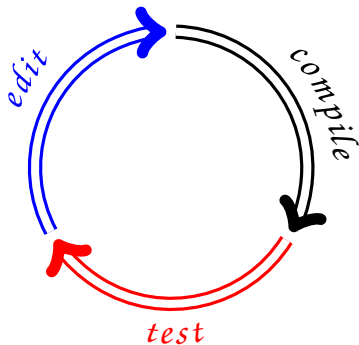


What is a compiler warning (as opposed to an error)?

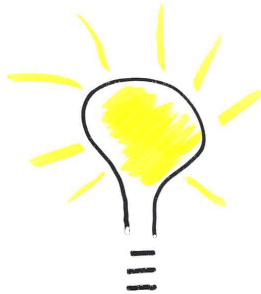
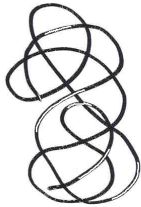
What you ever encountered a compiler warning issued by `javac`?

## Editing versus refactoring

# Program Development



At what point does planning and thinking come in?



1. design
2. experience
3. problem solving
4. pseudo code, flow charts
5. bring pencil and paper to lab