Developing Java Programs – BlueJ

```
/**
 * A class representing students for a simple database.
 * @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, years
     */
    public Student(String name, int years)
    {
        super(name, years);
        SID = "(unknown ID)";
    }
}
```
Developing Java Programs – Eclipse
Developing Java Programs – Emacs

```java
import java.util.Scanner;

public final class CopyText {
    public static void main (final String[] args) {
        final Scanner stdin = new Scanner(System.in);
        // Read the standard input 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
        }
    }

    public final class CopyText {
        public static void main (final String[] args) {
            final Scanner stdin = new Scanner(System.in);
            // Read the standard input stream as text and copy to standard out
            import java.util.Scanner;
        }
    }
}
```
• compile error
  • syntax error — Syntax.java
  • semantic error — Semantic.java
  • type error — Type.java

• 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 a executable program.

List of errors

• 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.
• 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 a 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`
$javac -X

auxiliaryclass a hidden auxiliary class used from other files.
cast use of unnecessary casts.
classfile issues related to classfile contents.
deprecation use of deprecated items.
divzero division by constant integer 0.
empty empty statement after if.
extports issues regarding module exports.
fallthrough falling through from a case of a switch statement.
finally finally clauses that do not terminate normally.
options issues relating to use of command line options.
overloads issues regarding method overloads.
overrides issues regarding method overrides.
path invalid path elements on the command line.
rawtypes use of raw types.
removal use of API that has been marked for removal.
serial Serializable classes with no serial version ID.
static accessing a static member using an instance.
try issues relating to use of try blocks.
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?
Indenting is very important; Many annoying white-space complaints
• MagicNumber

• [Checkstyle IllegalToken] “Use double instead of float”

• [Checkstyle IllegalToken] “Avoid typecasts”
Integer.parseInt("42");  // String to int
Integer.valueOf("42");  // String to Integer
Double.parseDouble("42");  // String to double
Double.valueOf(42);  // double [int] to Double [double, autounboxing]
Math.round(3.4D)  // double to long
Math.ceil(3.4D)  // double to double!
Math.floor(3.4D)  // double to double!
floorDiv (42,43)  // int,int -> int
    d = Double.valueOf (42);  // int to double [big overhead]

Java API doc Math
No good explicit function to convert a primitive integer to a primitive double, e.g., \texttt{Real(42)} in Ada, \texttt{fromIntegral(42)} in Haskell. A cast (implicit widening conversion) could be

\begin{verbatim}
double quotient = (double) 42 / 5;  // Avoid cast

double meaningOfLife = 42;
double quotient = meaningOfLife / 5;
\end{verbatim}
Thou shalt indent by three

(Four is perfectly reasonably, but we cannot check for three or four.)
Editing versus refactoring
At what point does planning and thinking come in?
1 design
2 experience
3 problem solving
4 pseudo code, flow charts
5 AFK; pencil and paper
• Expect bugs
• Keep modules small
• Limit interactions
• Develop code incrementally
• Solve an easier problem
• Consider a recursive solution
• Build tools where appropriate
• Resuse software when possible