Program Development

Program Development

© 31 January 2021
Developing Java Programs – BlueJ

```java
/**
 * A class representing students for a simple school management system.
 * @author Michael Kolling
 * @version 1.0, January 1999
 */
public 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, and ID.
     */
    public Student(String name, int year, String id) {
        super(name, year);
        SID = id;
    }
    String getRoom() {
        // Implementation...
    }
    void setRoom(String room) {
        // Implementation...
    }
    String toString() {
        // Implementation...
    }
}
```
Developing Java Programs – Eclipse
Developing Java Programs – Emacs

```java
public final class CopyText {
    public static void main(String[] args) {
        final Scanner stdin = new Scanner(System.in);
        System.out.println(stdin.hasNextLine());
        final String line = stdin.nextLine();
        System.out.println(line);
    }
}
```

```
H:\public_html\java\programs\io>javac -version
javac 1.6.0_01
H:\public_html\java\programs\io>javac CopyText.java
// CopyText.java -- read the standard input stream as text and copy to standard out
import java.util.Scanner;

public final class CopyText {
    public static void main(String[] args) {
        final Scanner stdin = new Scanner(System.in);
        System.out.println(stdin.hasNextLine());
        final String line = stdin.nextLine();
        System.out.println(line);
    }
}
H:\public_html\java\programs\io>javac CopyText < CopyText.java
// CopyText.java -- read the standard input stream as text and copy to standard out
import java.util.Scanner;

public final class CopyText {
    public static void main(String[] args) {
        final Scanner stdin = new Scanner(System.in);
        System.out.println(stdin.hasNextLine());
        final String line = stdin.nextLine();
        System.out.println(line);
    }
}
```
• 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
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., `Real(42)` in Ada, `fromIntegral(42)` in Haskell. A cast (implicit widening conversion) could be

```java
double quotient = (double) 42 / 5;  // Avoid cast
```

```java
double meaningOfLife = 42;
double quotient = meaningOfLife / 5;
```
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
CREATE  STRATEGIZE  POLISH