Session 1-2

Programming and Java

JDK/J2SE Versions

Version	Date
JDK Beta	1995
JDK 1.0	January 23, 1996[40]
JDK 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8 (LTS)	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11 (LTS)	September 25, 2018[41]
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020[42]
Java SE 16	March 16, 2021
Java SE 17 (LTS)	September 14, 2021
Java SE 18	March 2022

Source: https://en.wikipedia.org/wiki/Java_version_history

JDK Editions

- **Java Standard Edition (J2SE)**
  - J2SE can be used to develop client-side standalone applications or applets.
- **Java Enterprise Edition (J2EE)**
  - J2EE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- **Java Micro Edition (J2ME)**
  - J2ME can be used to develop applications for mobile and embedded devices devices such as cell phones.

Java IDE Tools

- Borland JBuilder
- NetBeans Open Source by Sun
- Sun ONE Studio by Sun MicroSystems
- Eclipse Open Source by IBM

A Simple Java Program

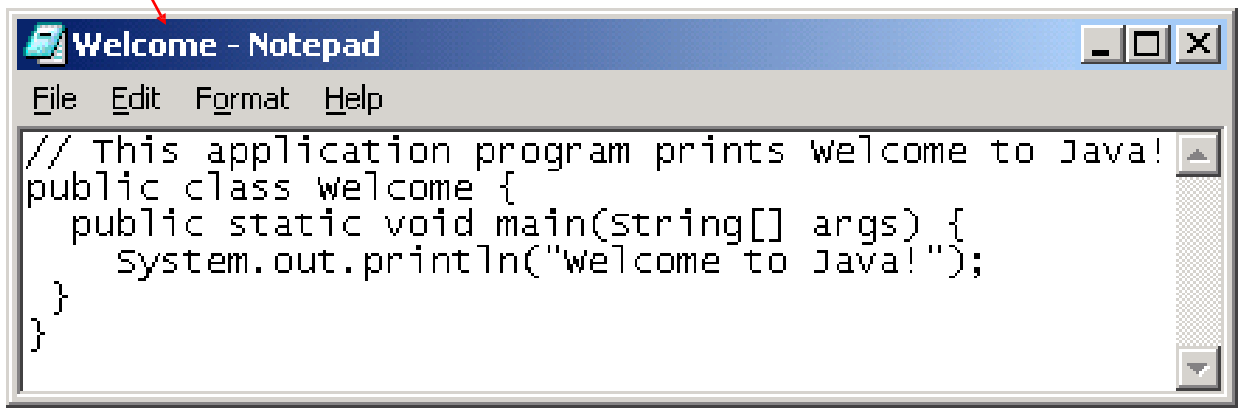
```
//This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Creating and Editing Using NotePad

To use NotePad, type

notepad Welcome.java

from the DOS prompt.

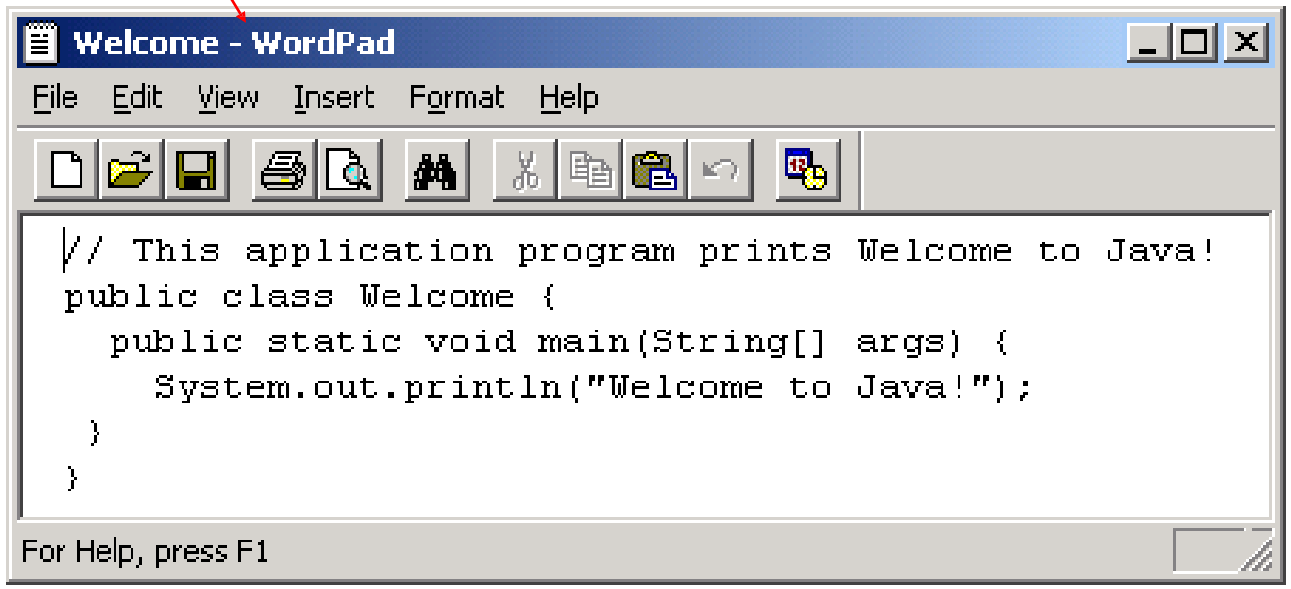


Creating and Editing Using WordPad

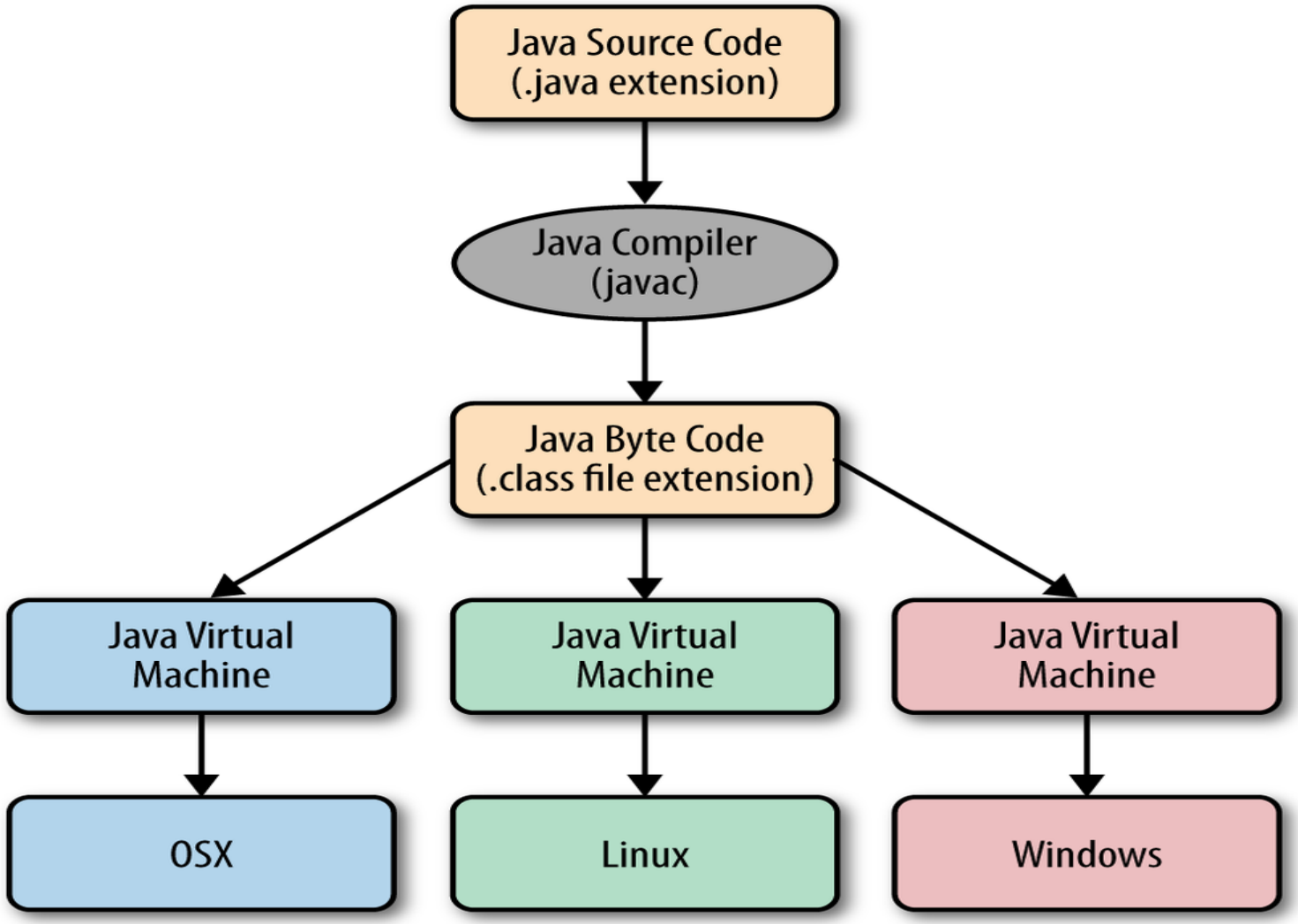
To use WordPad, type

write Welcome.java

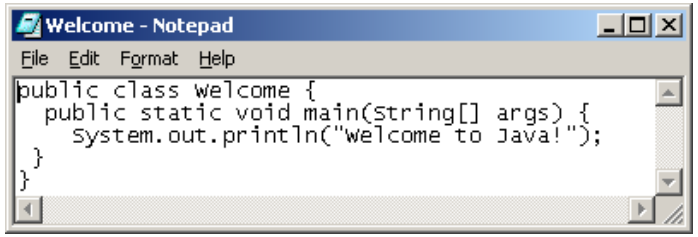
from the DOS prompt.

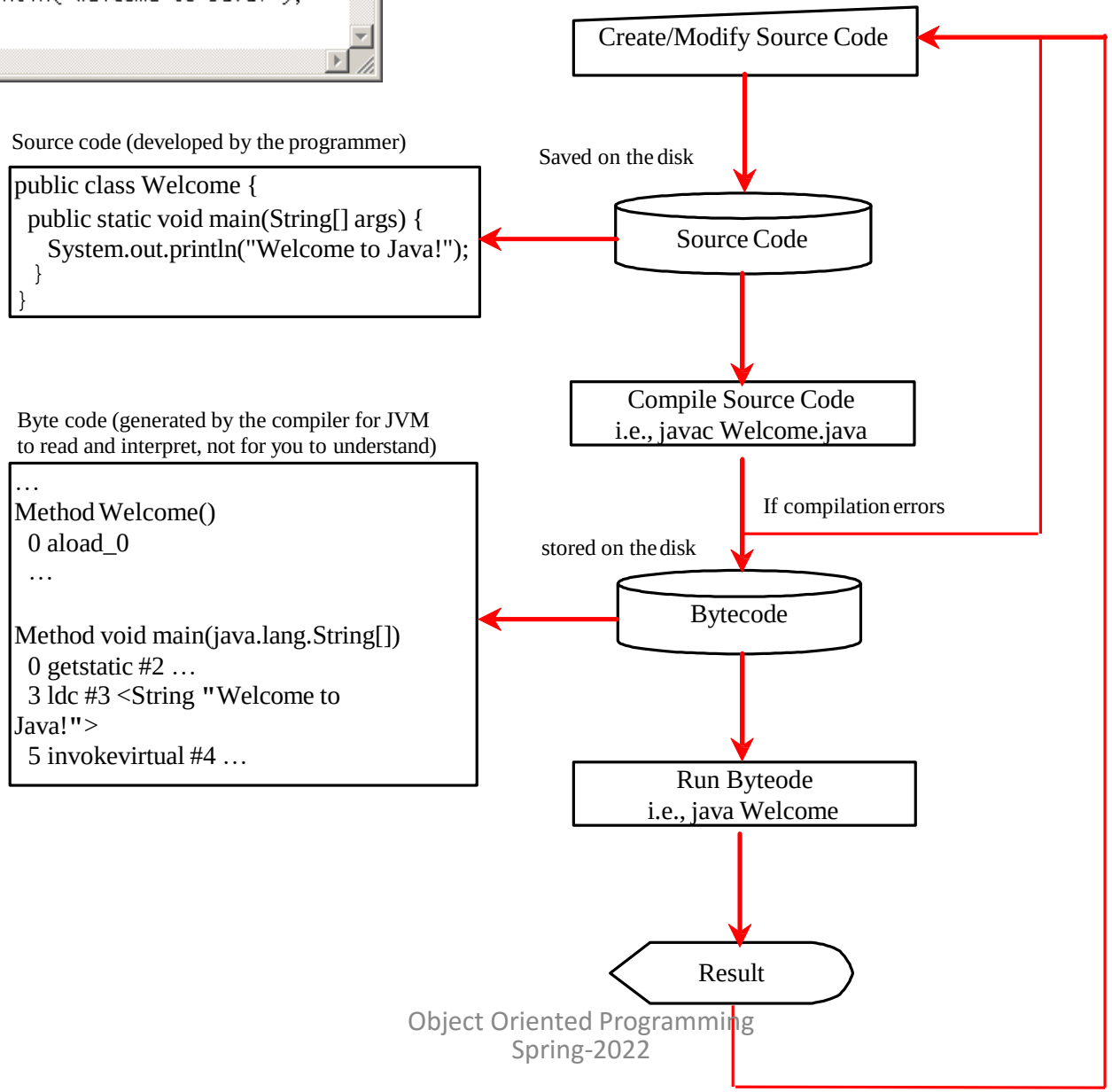


Java Compilation



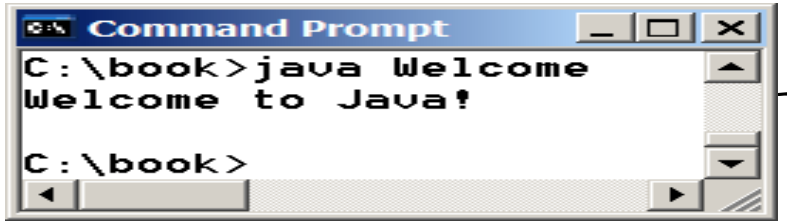
Creating, Compiling, and Running Programs





Trace a Program Execution

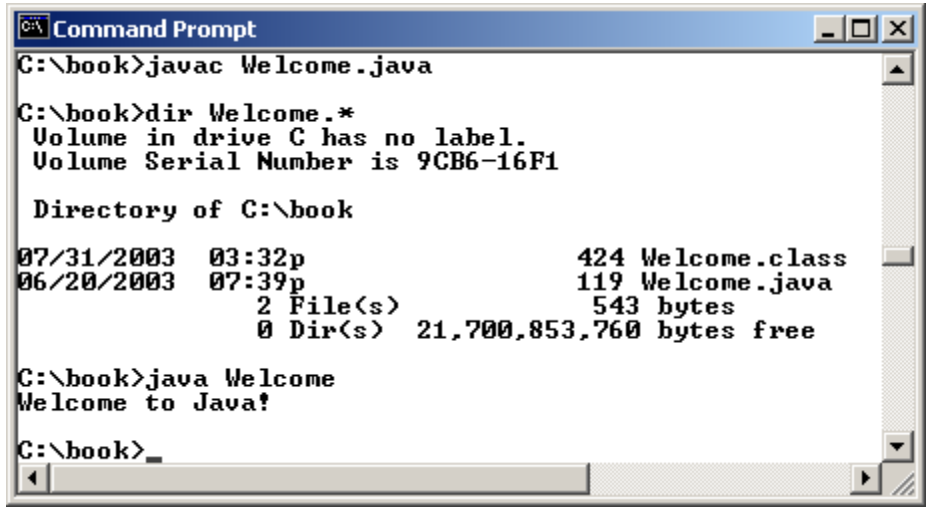
```
//This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```



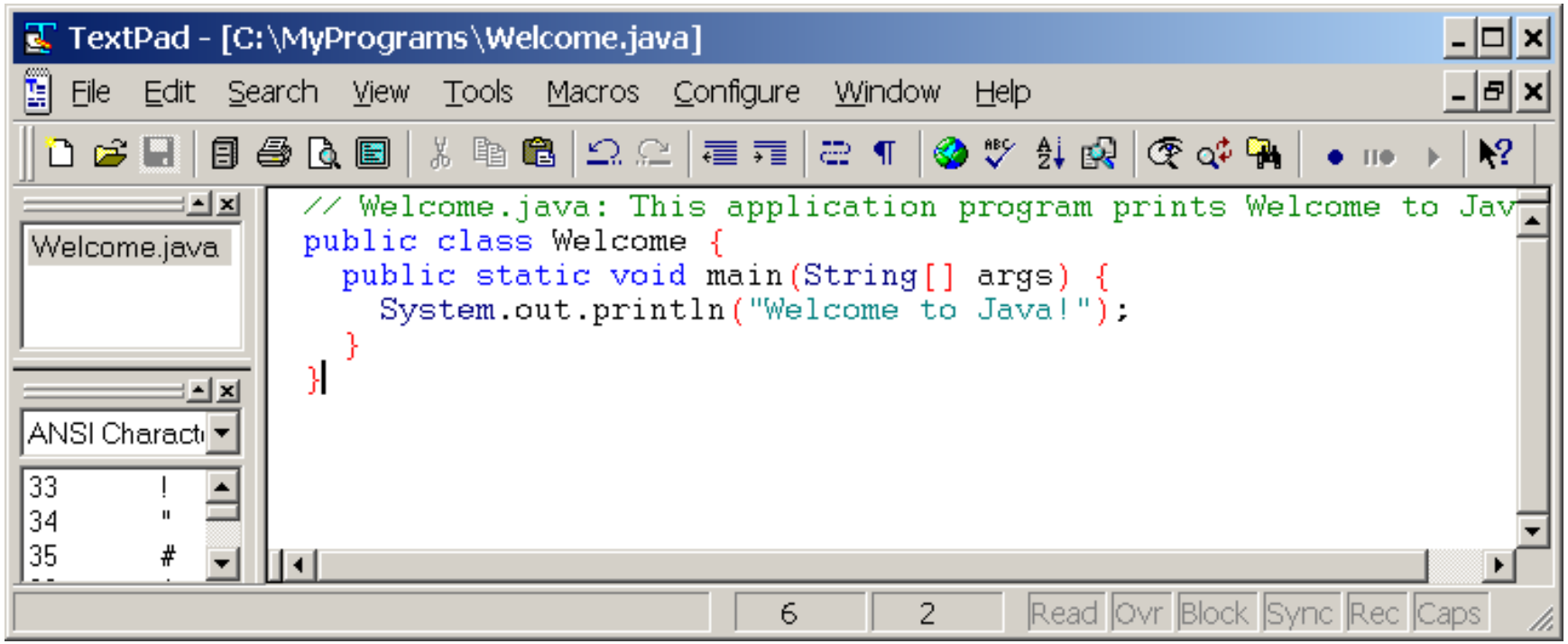
print a message to the console

Compiling and Running Java from the Command Window

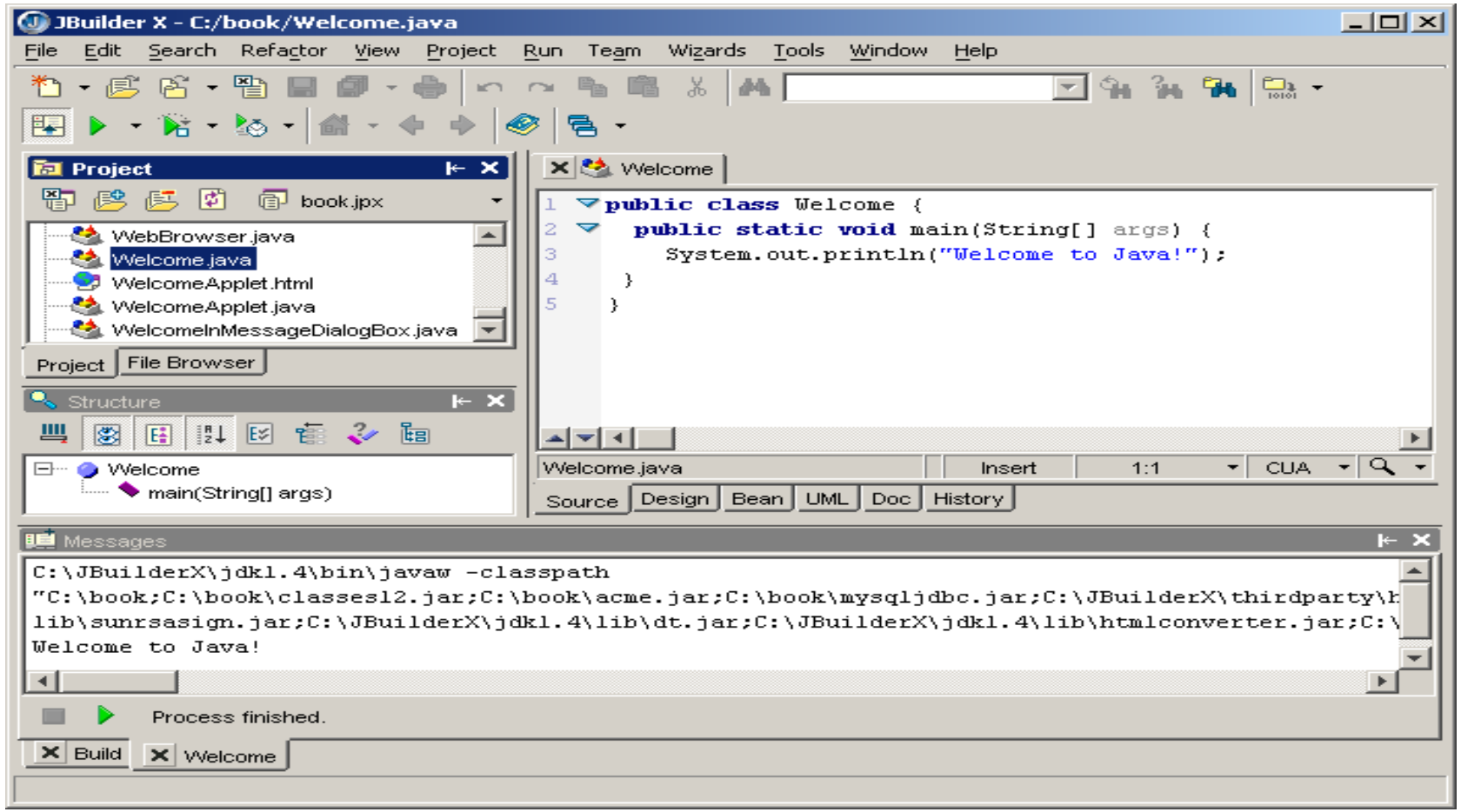
- Set path to JDK bin directory
  - set path=c:\Program Files\java\jdk1.8.0\bin
- Compile
  - javac Welcome.java
- Run
  - java Welcome



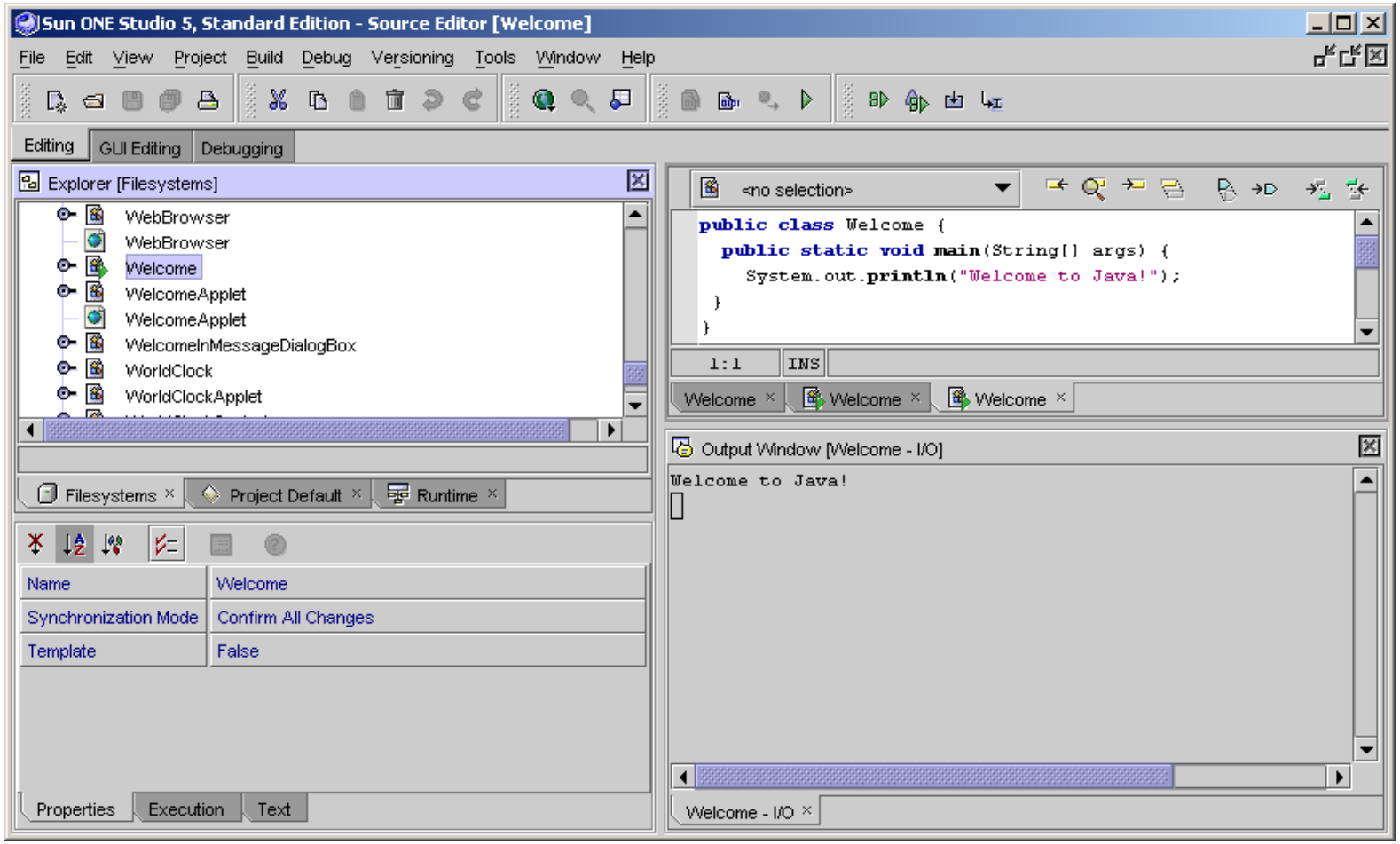
Compiling and Running Java from TextPad



Compiling and Running Java from JBuilder



Compiling and Running Java from NetBeans



Reading input in JAVA

READING INPUT FROM THE input

- Previously, we have seen the standard output device is a console window and the *System.out* object is associated with the standard output device.
- The standard input device is normally the computer input.
- The Java API has an object *System.in* which is associated with the standard input device.

READING INPUT FROM THE input
The *Scanner* Class

- We will use the *System.in* object in conjunction with the *Scanner* class to read input data from the input.
- The *Scanner* class has methods that can be used to read input and format it as either a value of a primitive data type or a *String*.

READING INPUT FROM THE input
The *Scanner* Class

- To use the *Scanner* class in our program we must put the following statement near the top of our file, **before any class definition**:

```
import java.util.Scanner;
```

This statement tells the compiler where to find the *Scanner* class in the Java API.

READING INPUT FROM THE input
The *Scanner* Class

- You must also create a *Scanner* object and connect it to the *System.in* object. You can do this with a statement like the following:

```
Scanner input= new Scanner(System.in);
```

Create this object inside your *main* method before you attempt to read anything from the input.

READING INPUT FROM THE input
The *Scanner* Class

- The words *Scanner input* declare a variable named *input* of type *Scanner*. This variable will reference an object of the *Scanner* class.

```
Scanner input= new Scanner(System.in);
```

You could have chosen any name you wanted for the variable, but *input* is a good one since you are going to use it to access the keyboard i.e. an input device.

READING INPUT FROM THE input
The *Scanner* Class

```
Scanner input= new Scanner(System.in);
```

- The `new` key word is used to create an object in memory.
- In the statement above we are creating an object of the *Scanner* class.
- Inside the parentheses, we have *System.in*. Here we are saying that we want the object we are creating to be connected with the *System.in* object, which again is associated with the keyboard.
- We are assigning the address of the object created using the `new` operator to our variable named *input*, so *input* now references the object we have linked with the actual keyboard.

READING INPUT FROM THE input
The *Scanner* Class

- Every object created from the *Scanner* class has methods that read a string of characters entered at the input, convert them to a specified type, and return the converted value. This value can be stored in a variable of compatible type.

READING INPUT FROM THE input
The *Scanner* Class

For example, the code below could be used to read an integer entered at the input and store it in an integer variable named *age*.

```
int age;

System.out.print("Enter your age: ");
age = input.nextInt( );
```

- The *nextInt( )* method formats the characters entered by the user as an `int` and returns the integer value.
- The integer value is assigned to the variable named *age*.

Method	Example and Description

nextByte	**Example Usage:**
	`byte x;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter a byte value: ");`
	`x = keyboard.nextByte();`
	Description: Returns input as a byte.

nextDouble	**Example Usage:**
	`double number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter a double value: ");`
	`number = keyboard.nextDouble();`
	Description: Returns input as a double.

nextFloat	**Example Usage:**
	`float number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter a float value: ");`
	`number = keyboard.nextFloat();`
	Description: Returns input as a float.

nextInt	**Example Usage:**
	`int number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter an integer value: ");`
	`number = keyboard.nextInt();`
	Description: Returns input as an int.

nextLine	**Example Usage:**
	`String name;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter your name: ");`
	`name = keyboard.nextLine();`
	Description: Returns input as a String.

nextLong	**Example Usage:**
	`long number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter a long value: ");`
	`number = keyboard.nextLong();`
	Description: Returns input as a long.

nextShort	**Example Usage:**
	`short number;`
	`Scanner keyboard = new Scanner(System.in);`
	`System.out.print("Enter a short value: ");`
	`number = keyboard.nextShort();`
	Description: Returns input as a short.

READING INPUT FROM THE input
The *Scanner* Class

- We can use the *nextLine* method of a *Scanner* object to read a string of characters entered at the keyboard.

  Example:

  To get the user's first name we could write:

```
String firstName;

System.out.print("Enter your first name: ");
firstName = input.nextLine( );
```

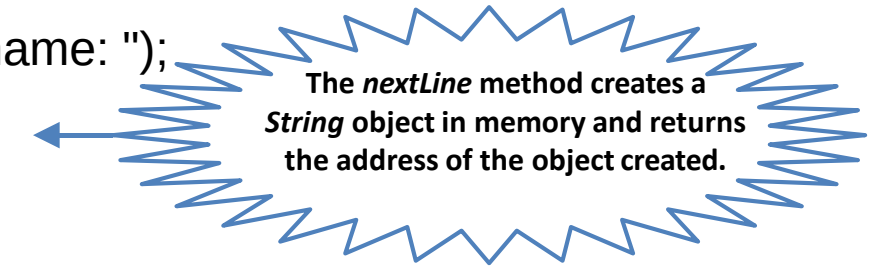
READING INPUT FROM THE input
The *Scanner* Class

- The *nextLine* method creates a *String* object in memory that contains the sequence of characters entered at the keyboard before the **Enter** key is pressed and returns the address of this object.

Below we are assigning the address of the object created by the *nextLine* method to the *String* reference variable named *firstName*.

```
String firstName;


System.out.print("Enter your first name: ");
firstName = input.nextLine( );
```



READING INPUT FROM THE input
The *Scanner* Class

- The *Scanner* class does not have a method for reading a single character.
- In the text, they suggest using the *Scanner* classes *nextLine* method to read the character as a string, and then using the *String* classes *charAt* method to extract the first character from the string. Remember, the first character is at index 0.

READING A SINGLE CHARACTER ENTERED AT THE input

Example:

```
String stringInitial;
char initial;

System.out.print("Enter your middle initial " );
stringInitial = input.nextLine( );
initial =  stringInitial.charAt(0);
```

Java Programming Constructs

Java Identifiers

- Identifiers
  - Used to name local variables
  - Names of attributes
  - Names of classes

- Primitive Data Types Available in Java (size in bytes)
  - byte (1), -128 to 127
  - short (2), -32768 to 32767
  - int (4), -2147483648 to 2147483647
  - long (8), -9223372036854775808 to 9223372036854775807
  - float (4), -3.4E38 to 3.4E38, 7 digit precision
  - double (8), -1.7E308 to 1.7E308, 17 digits precision
  - char (2), unicode characters
  - boolean (true, false), discrete values

Java Identifiers

- Naming Rules
  - Must start with a letter
  - After first letter, can consist of letters, digits (0,1,…,9)
  - The underscore “_” and the dollar sign “$” are considered letters
- Variables
  - All variables must be declared in Java
  - Can be declared almost anywhere (scope rules apply)
  - Variables have default initialization values
    - Integers: 0
    - Reals: 0.0
    - Boolean: False
  - Variables can be initialized in the declaration

Java Identifiers

- Example Declarations

```
int speed;                          // integer, defaults to 0
int speed = 100;                    // integer, init to 100
long distance = 3000000000L;        // "L" needed for a long
float delta = 25.67f;               // "f" needed for a float
double delta = 25.67;               // Defaults to double
double bigDelta = 67.8E200d;        // "d" is optional here
boolean status;                     // defaults to "false"
boolean status = true;
```

- Potential Problems (for the C/C++ crew)

```
long double delta = 3.67E204;       // No "long double" in
                                    Java
unsigned int = 4025890231;          // No unsigned ints in
                                    Java
```

Java Types

- Arrays

```
int[] numbers = new int[n]
      // Array of integers, size is n
```

  - size can be computed at run time, but can't be changed
  - allocated on heap (thus enabling run time size allocation)
  - invalid array accesses detected at run time (e.g. numbers[6];)
  - numbers.length; // read only variable specifying length of array
  - reference semantics

```
int[] winning_numbers;
winning_numbers = numbers; // refer to same array
numbers[0] = 13;           // changes both
```

Java Types

- Strings

```
String message = "Error " + errnum;
```

  - strings are immutable – can't be changed, although variables can be changed (and old string left for garbage collection)
  - message = "Next error " + errnum2;
  - use StringBuffer to edit strings

```
StringBuffer buf = new StringBuffer(greeting);
buf.setCharAt( 4, '?');
greeting = buf.toString();
```

Java Types

- Strings
  - String comparison

```
if (greeting == "hello" ) ….
      // error, compares location only
if ( greeting.equals("hello")) …. // OK
string1.compareTo(string2)
      // negative if string1 < string 2;
      // zero when equal,
      // positive if string1 > string2
string1.substring(2, 6);
   // return substring between position 2
 and 5
```

Java Statements

- Assignments
  - General Format: variable = expression ;

    Where variable is a previously declared identifier and expression is a valid combo of identifiers, operators, and method (a.k.a. procedure or function) calls

  - Shortcuts:

```
var *= expr ;  // Equivalent to var = var * (expr);
var /= expr ;  // Equivalent to var = var / (expr);
var += expr ;  // Equivalent to var = var + (expr);
var -= expr ;  // Equivalent to var = var - (expr);
var %= expr ;  // Equivalent to var = var % (expr);
var++;         // Equivalent to var = var + 1;
var--;         // Equivalent to var = var - 1;
```

Java Conditional Constructs

- “if” Statements
  - if with code block

```
if (boolean_expr)
{
   statements
}
```

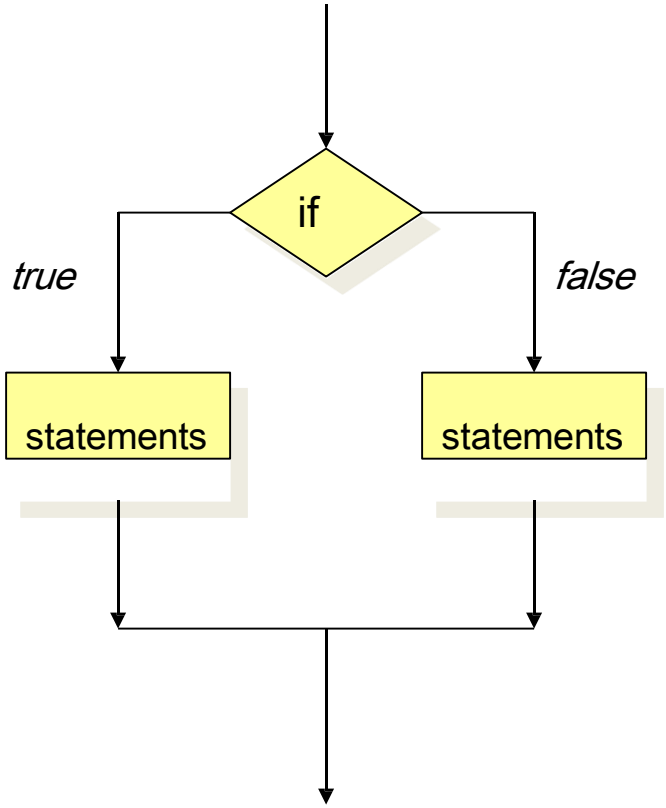
  - if with single statement

```
if (boolean_expr)
   statement;
```

Java Conditional Constructs

- if" Statements (Continued)
  - if-else

```
if (boolean_expr)
{
   statements for true
}
else
{
   statements for false
}
```



Java Conditional Constructs

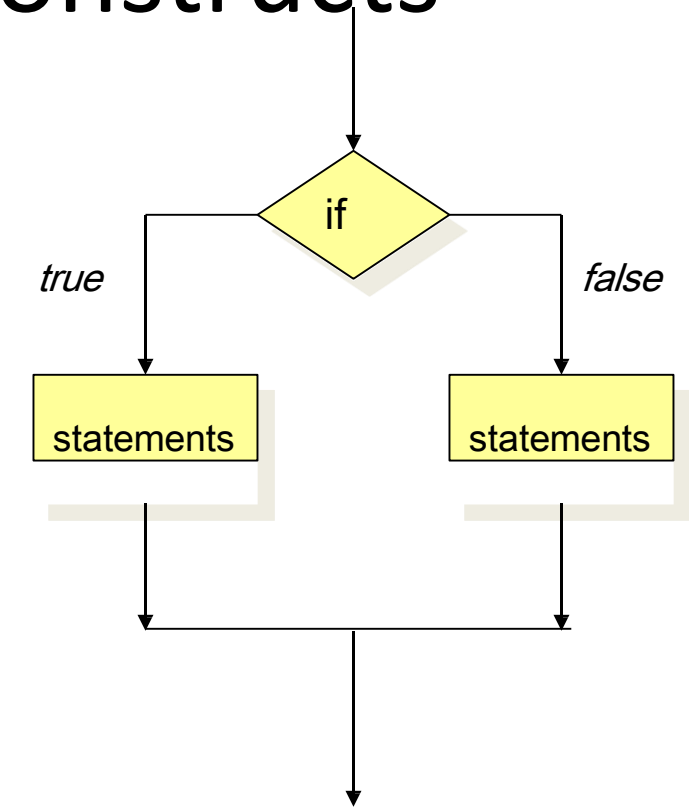
- Boolean Expressions
  - Boolean expressions use conditional operators such that they result in a value of true or false
  - Conditional Operators (Not by order of precedence)

Operator	Operation
== or !=	Equality, not equal
> or <	Greater than, less than
>= or <=	Greater than or equal, less than or equal
!	Unary negation (NOT)
& or &&	Evaluation AND, short circuit AND
| or ||	Evaluation OR, short circuit OR

Java Conditional Constructs

- if-else" Statement Example

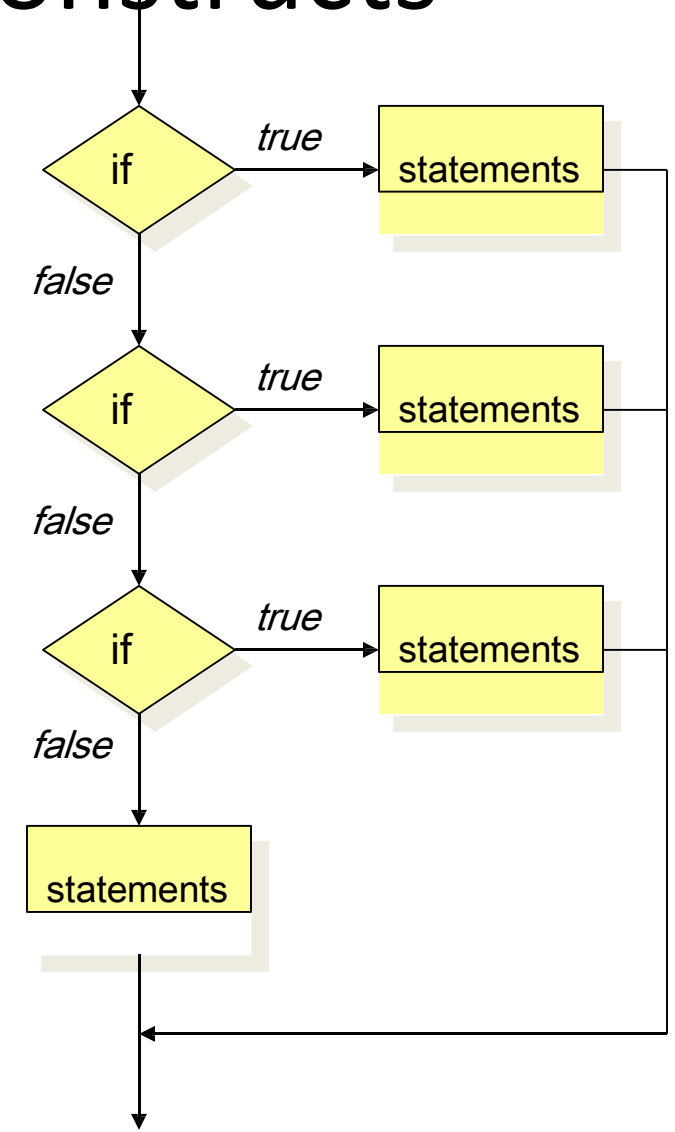
```
class Example
{
   static public void main(String args[])
   {
        // A very contrived example
        int i1 = 1, i2 = 2;
        System.out.print("Result: ");
        if (i1 > i2)
        {
            System.out.println("i1 > i2");
        }
        else
        {
            System.out.println("i2 >= i1");
        }
   }
}
```



Java Conditional Constructs

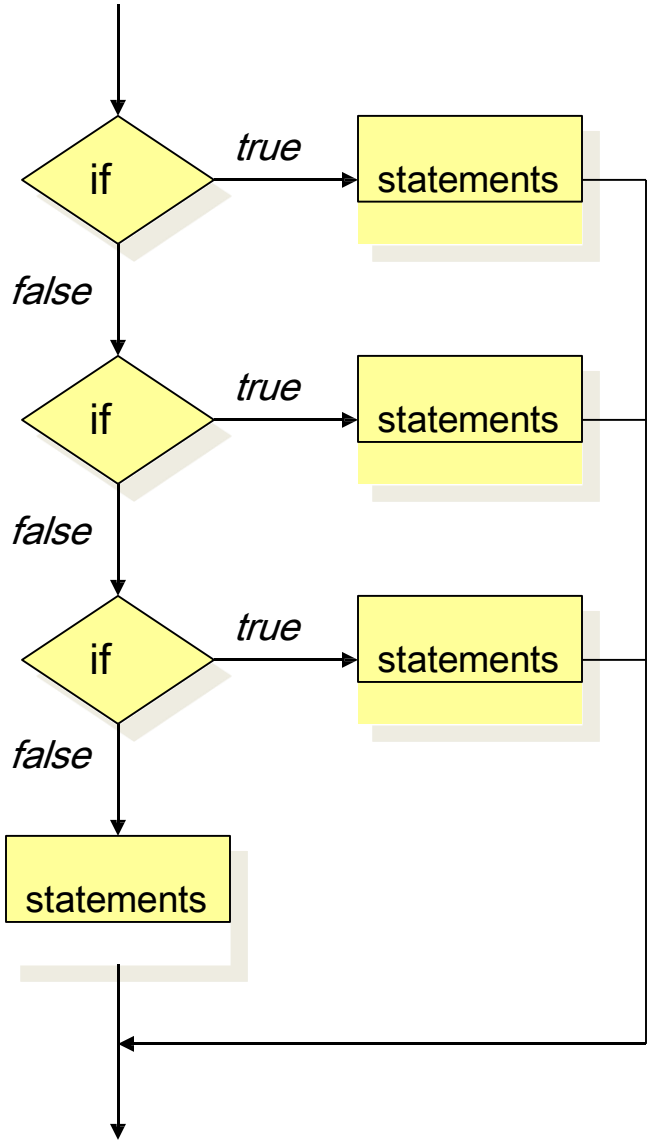
- The Switch Statement

```
switch (integer_expression)
{
    case int_value_1:
        statements
        break;
    case int_value_2:
        statements
        break;
    …
    case int_value_n:
        statements
        break;
    default:
        statements
}
```



- Don’t forget the “break”

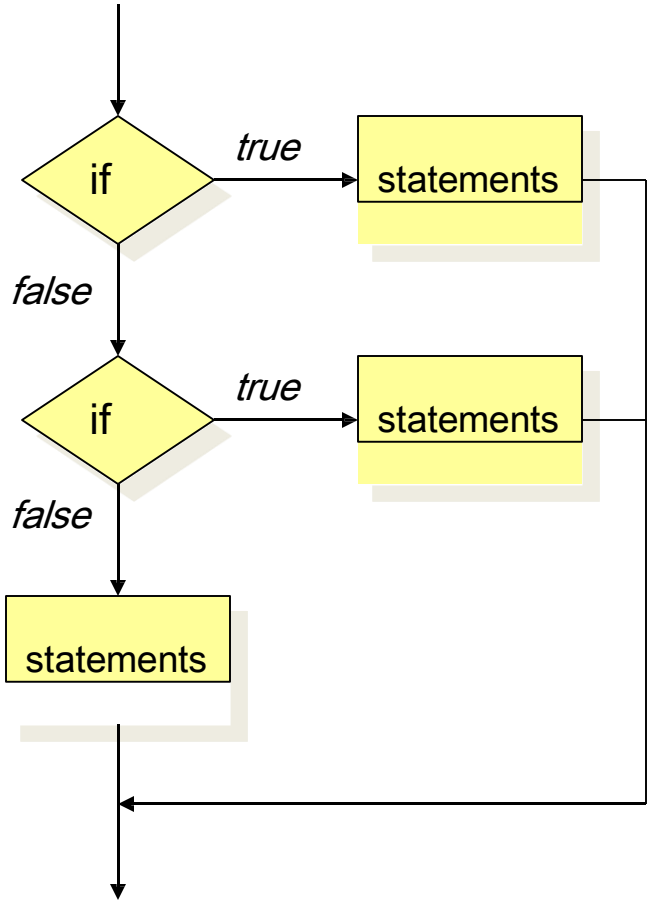
```
switch (integer_expression)
{
    case int_value_1:
        statements
        // No break!
    case int_value_2:
        statements
        break;
    …
    case int_value_n:
        statements
        break;
    default:
        statements
}
```



Java Conditional Constructs

- Example

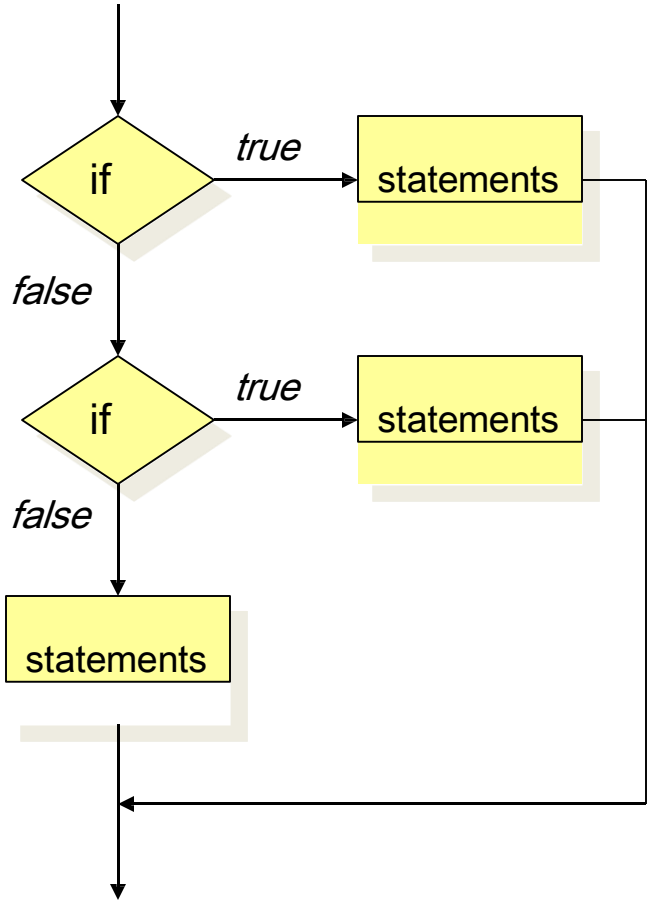
```
int n = 5;
switch (n)
{
    case 1:
        n = n + 1;
        break;
    case 5:
        n = n + 2;
        break;
    default:
        n = n - 1;
}
```



Java Conditional Constructs

- Example

```
char c = 'b';
int n = 0;
switch (c)
{
        case 'a':
        n = n + 1;
        break;
        case 'b':
        n = n + 2;
        break;
        default:
         n = n - 1;
}
```



Java Looping Constructs

- while loop
  - Exit condition evaluated at top
- do loop
  - Exit condition evaluated at bottom
- for loop
  - Exit condition evaluated at top
  - Includes a initialization statements
  - Includes a update statements for each iteration

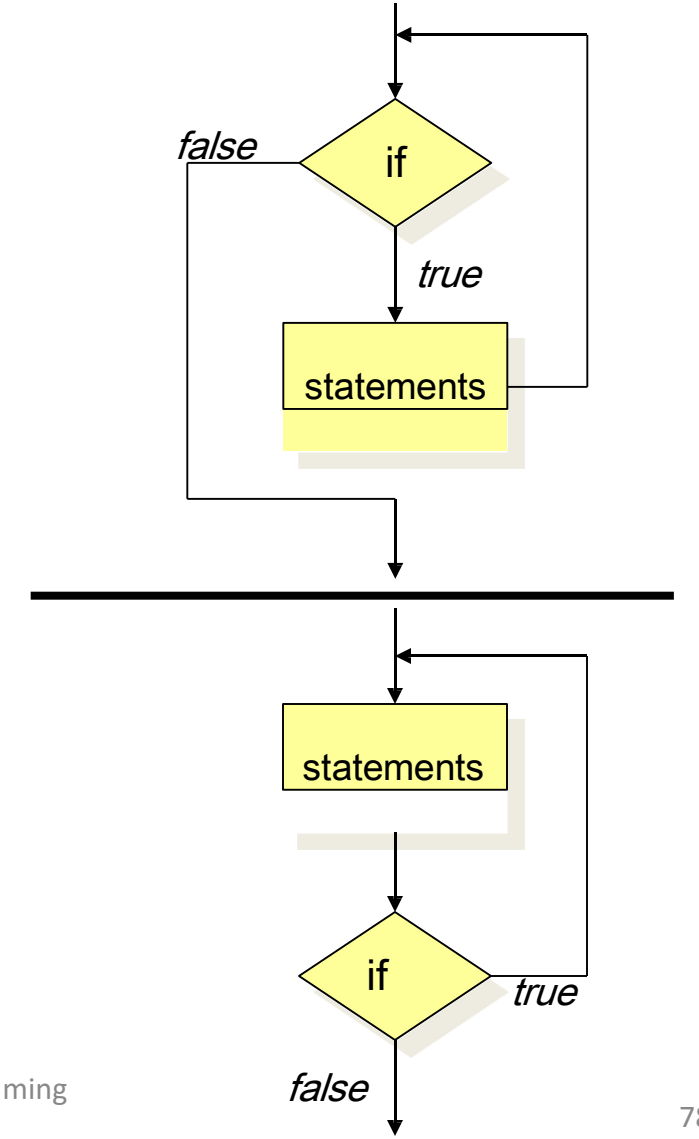
Java Looping Constructs

- while loop

```
while (boolean_expr)
{
    statements
}
```

- do loop

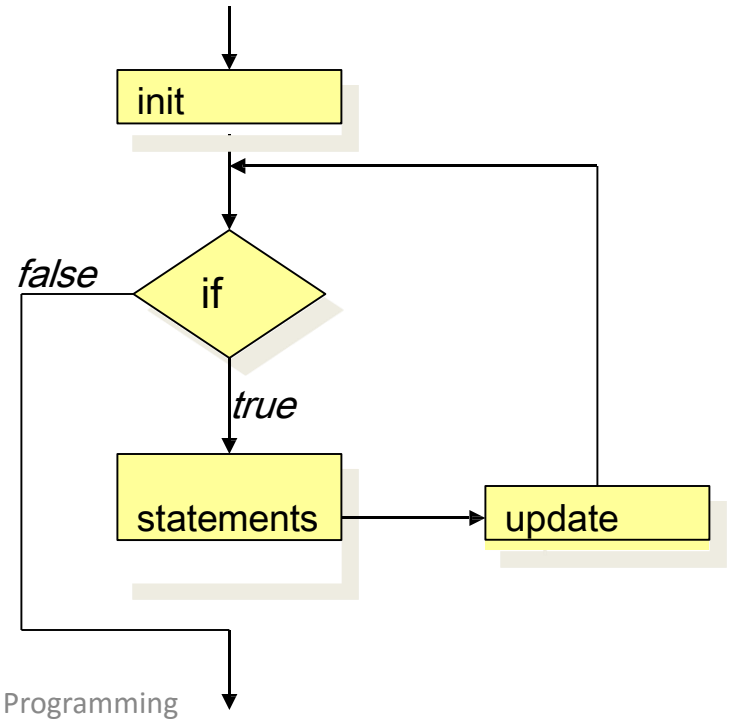
```
do
{
    statements
}
while (boolean_expr)
```



Java Looping Constructs

- for loop

```
for (init_stmnt; bool_expr; update_stmnt)
{
 statements
}
```



```
class Example
{
    static public void main(String args[])
    {
       int i = 0;
       System.out.println("while loop");
       while (i < 10)
       {
          System.out.println(i);
          i++;
       }
       System.out.println("do loop");
       do
       {
          System.out.println(i);
          i--;
       }
       while (i > 0);
       System.out.println("for loop");
       for (i = 0; i < 10; i++)
       {
          System.out.println(i);
       }
    } // End main
} // End Example
```