

FIT ICT Software Development

Lecture 3

Lecturer : Charles Tyner

RECAP

Declaring Variables in Java

- The procedure of creating **named locations** in the computer's memory that will contain values while a program is running;
- These **named locations** are called **variables** because their values are allowed to *vary* over the life of the program.

To **create a variable in your program you must:**

- Give that variable a **name** (of your choice);
- Decide which **data type** in the language best reflects the kind of values you wish to store in the variable.

Choosing a suitable Data Type

Data can be classified as Numeric Four Java types can be used to hold integers (**byte**, **short**, **int** and **long**).

Two Java types that can be use to hold real numbers (**double** and **float**).

the **int** type is often chosen to store integers

the **double** type is often chosen to store real numbers

The difference among these types is the range of values that they can keep

The data type char is used to hold characters.

Once name and type decided upon, the variable is declared as follows:

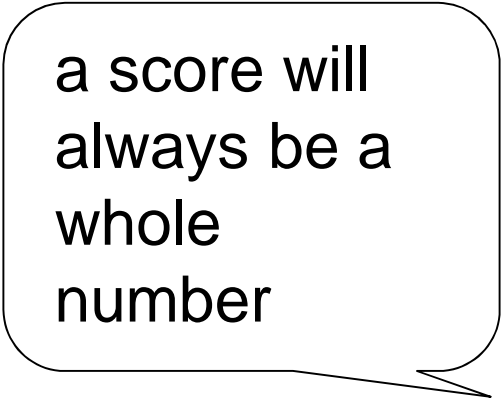
```
dataType variableName ;
```

The scalar types of Java

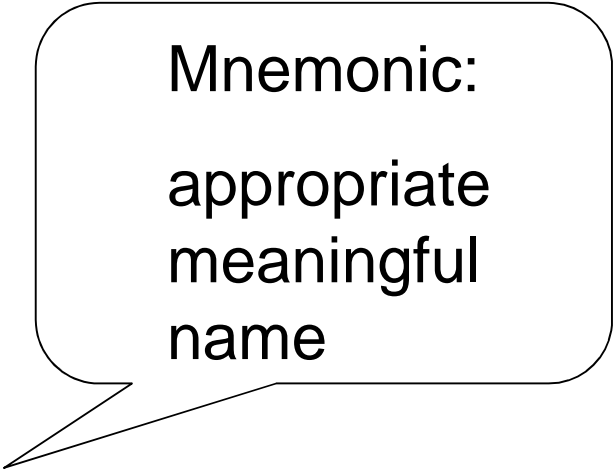
<u>The scalar types of Java</u>		
<u>Java type</u>	<u>Allows for</u>	<u>Range of values</u>
byte	very small integers	-128 to 127
short	small integers	-32768 to 32767
int	big integers	-2147483648 to 2147483647
long	very big integers	-9223372036854775808 to 9223372036854775807
float	real numbers	+/- $1.4 * 10^{-45}$ to $3.4 * 10^{38}$
double	very big real numbers	+/- $4.9 * 10^{-324}$ to $1.8 * 10^{308}$
char	characters	Unicode character set
boolean	true or false	not applicable

Declaring a variable: An example

Let's create a variable to keep a player's score in a computer game.

A speech bubble with a tail pointing towards the bottom right.

a score will
always be a
whole
number

A speech bubble with a tail pointing towards the bottom left.

Mnemonic:

appropriate
meaningful
name

int score ;

Declaring variables of the same type

Several variables can be declared on a **single line** if they are **all of the same type**.

Assume that there are ghosts in the house that hit out at the player; the number of times a player gets hit by a ghost can also be recorded.

```
int score, hits; // both the same type  
char level ; // different type
```

Reserved Words

High Level Languages, including Java, are usually defined in terms of a set of **reserved** words. The Java programming language is defined in terms of around 40 reserved words, which make up most of the basic command vocabulary of the language.



Reserved words in Java

exit
break
void
if
main
case
end
system
double
do
else
(etc...)

Note:

These reserved words cannot be redefined for other uses: i.e. they may not be used as identifiers for variables. (Please refer to Appendix C in Deitel and Deitel for full List of Keywords and Reserved Words)

Assignments in Java

Assignments in Java

Assignments allow **values to be put** into **variables**.

Written in Java with the use of the equality symbol (=), known as **the assignment operator**.

Simple assignments take the following form:

variableName = value;

```
score = 0;
```

Initializing variables

You may **combine** the **assignment statement** with a **variable declaration** to put an initial value into a variable as follows:

```
int score = 0;
```

Note, the following declaration will **not** compile in Java:

```
int score = 2.5 ;
```

This will not compile because 2.5 is a **double** value

Putting values into character variables

When assigning a value to a character variable, you must **enclose the value in single quotes**.

for example

set the initial difficulty level to A

```
char level = 'A';
```

Re-assigning variables

Remember: you need to declare a variable only once.

You can then assign to it as many times as you like.

```
char level = 'A';  
level = 'B';
```

Re-assigning variables

Whenever a value is placed in a memory location, this value replaces the previous value in that location.

The previous value is therefore destroyed!

Note on Creating Constants

Constants are data items whose values **do not change**.

For example:

- the maximum score in an exam (100);
- the number of hours in a day (24);
- the mathematical value of π (3.1417).

Constants are declared much like variables except

- they are **preceded** by the keyword **final**
- they are always **initialised to their fixed value**.

Creating Constants

```
final int HOURS = 24;
```

Any attempt to change this value later on will result in a compile error. For Example

```
final int HOURS = 24; // create constant  
HOURS = 12;           // will not compile
```

The above code will result in a **compile error**.

Arithmetic Operators

Arithmetic Operators

Java has the **four familiar arithmetic operators**, plus a remainder operator for this purpose.

<u>The arithmetic operators of Java</u>	
<u>Operation</u>	<u>Java operator</u>
addition	+
subtraction	-
multiplication	*
division	/
remainder	%

Note on expressions in Java

Right-hand side of an assignment statement can itself contain variable names.

```
double price, tax, cost;  
price = 500;  
tax = 17.5;  
cost = price * (1 + tax/100) ;
```

**Value is
587.5**

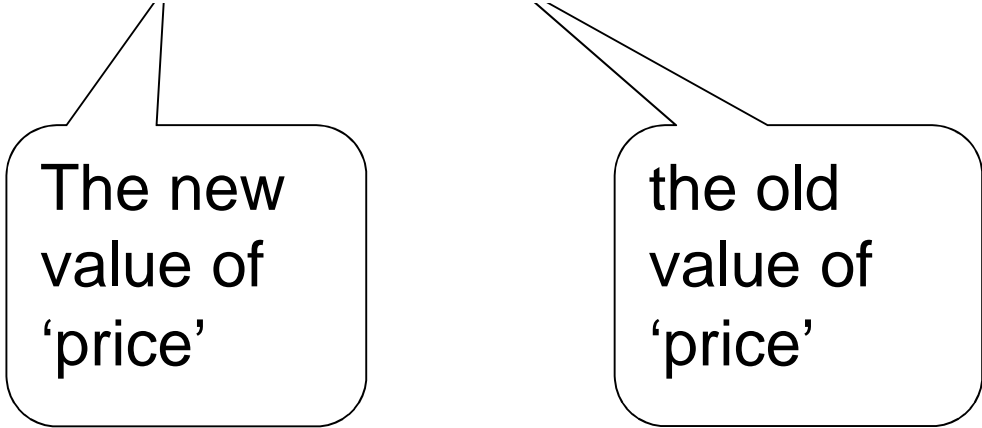
**Value is
500**

**Value is
17.5**

More expressions

Nothing to stop you using the name of the variable you are assigning to in the expression itself.

```
price = price * (1 + tax/100) ;
```



The new
value of
'price'

The diagram illustrates the execution of the code line 'price = price * (1 + tax/100) ;'. It features two callout boxes. The left box, labeled 'The new value of 'price'', has a pointer line connecting it to the 'price' variable in the assignment statement. The right box, labeled 'the old value of 'price'', has a pointer line connecting it to the 'price' variable in the expression part of the assignment statement.

the old
value of
'price'

This Session

- Introduce Program Design:
Algorithms, Pseudocode and Activity Diagrams
- Introduce Control Structures:
Sequence and Selection
- Use **If Statements** to make choices in a program
- Expand if statements to include **if/else statements**
- Use **switch statements**

- **Before** writing a program, it is essential:
 - to **understand the problem**
 - understand the **type of building blocks** that are available and employ proven program construction principles i.e. structured programming.

Algorithms

- Any computing problem can be solved by executing a series of actions in a **specific** order.
- A **procedure** for solving a problem in terms of:
 - the **actions** to be executed and
 - the **order** in which these actions are to be executed is called an **algorithm**.

Algorithms

- The “rise-and-shine algorithm” followed by one executive for getting out of bed and going to work:
(1) Get out of bed; (2) take off pajamas; (3) take a shower; (4) get dressed; (5) eat breakfast; (6) carpool to work.
- Suppose that the same steps are performed in a slightly different order:
(1) Get out of bed; (2) take off pajamas; (3) get dressed; (4) take a shower; (5) eat breakfast; (6) carpool to work.
- Specifying the order in which statements (actions) execute in a program is called program control

Building Blocks

Pseudocode

- **Pseudocode** is an informal language that helps you develop algorithms without having to worry about the strict details of Java language syntax.
- Particularly useful for developing algorithms that will be converted to structured portions of Java programs.
- Pseudo code is an **artificial** and **informal** language that helps programmers **develop algorithms**.
- Pseudo code is similar to everyday English.

Building Blocks

Pseudocode

- It is user friendly but is **NOT** a programming language.
- It helps the programmer “think out” a program before attempting to write it in a computer language such as Java.
- A carefully prepared pseudo code program may be **converted easily** to a corresponding Java program e.g. input, output or calculations

Pseudo code example

BEGIN

*DECLARE variables for number 1
number 2 and sum*

INITIALISE variables

PROMPT user to input values

SET sum to number 1 + number 2

DISPLAY sum

END

Building Blocks

Activity Diagram

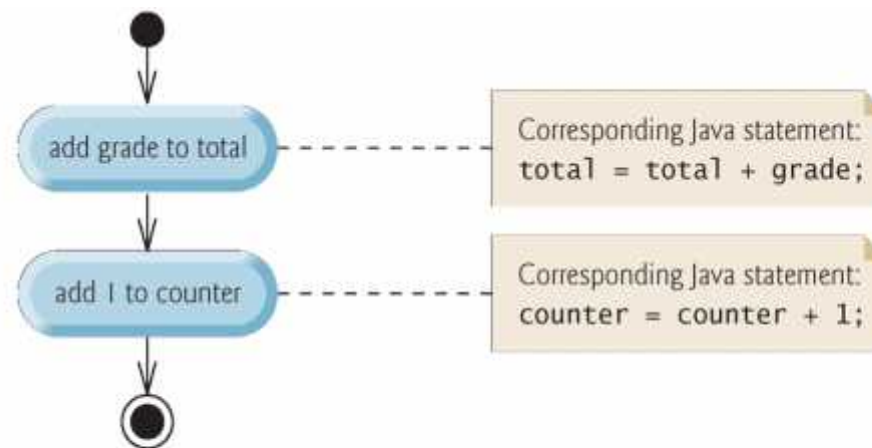


Fig. 3.1 | Sequence structure activity diagram.

Building Blocks

Activity Diagram

- UML activity diagram
- Models the **workflow** (also called the **activity**) of a portion of a software system.
- May include a portion of an algorithm.
- Composed of symbols
 - **action-state symbols** (rectangles with their left and right sides replaced with outward arcs)
 - **diamonds**
 - **small circles**
- Symbols connected by **transition arrows**, which represent the flow of the activity—the order in which the actions should occur.
- Help you develop and represent algorithms.
- Clearly show how control structures operate.

Building Blocks

Activity Diagram

- **Action states** represent actions to perform.
- Each contains an **action expression** that specifies a particular action to perform.
- Arrows represent **transitions** (order in which the actions represented by the action states occur).
- **Solid circle** at the top represents the **initial state**—the beginning of the workflow before the program performs the modeled actions.
- **Solid circle surrounded by a hollow circle** at the bottom represents the **final state**—the end of the workflow after the program performs its actions.

Control Structures

Sequence

All programs are written in terms of three control structures:

- Sequence structure
- Selection structure
- Repetition structure.

Sequence

- Every program is formed by combining the sequence statement, selection statements (three types) and repetition statements (three types) as appropriate for the algorithm the program implements.
- Can model each control statement as an activity diagram.

Sequence Control Structure

- Built into Java.
- Unless directed otherwise, the computer executes Java statements one after the other in the order in which they're written.
- The [activity diagram](#) previous illustrates a typical sequence structure in which two calculations are performed in order.
- Java lets you have as many actions as you want in a sequence structure.
- Anywhere a single action may be placed, we may place several actions in sequence.

Sequence

- All programs up until now have been executed in **sequence** i.e. **one instruction after the next**, from top to bottom.
- At first sight this may not seem a problem. Why would you need to execute your instructions in any other order?
- There are numerous instances when this order is **too restrictive** and you will want **more control** over the order in which the instructions are executed.
- This is called **transfer of control**.

Selection Control Structure

Programs often need to make *choices*.

for example

a program processing requests for airline tickets could have the **following choices** to make:

- display the price of the seats requested;
- display a list of alternative flights;
- display a message saying that no flights are available to that destination.

Selection allows such choices to be made in programs.

Selection: a simple example

Consider the following code fragment

```
System.out.println("How old are you?");  
age = input.nextInt();  
System.out.println("Hello Junior!");  
System.out.println("Enjoy your trip");
```

Two program interactions

How old are you?

10

Hello Junior!

Enjoy your trip



**first program
interaction**

How old are you?

65

Hello Junior!

Enjoy your trip



**second program
interaction**

A first attempt at fixing the program

```
System.out.println("How old are you?");  
age = input.nextInt();  
IF age is that of a child  
BEGIN  
    System.out.println("Hello Junior!");  
END  
System.out.println("Enjoy your trip");
```

Implementing this selection in Java

This form of selection in Java involves the use of a **boolean condition**.

```
if ( /* boolean condition goes here */ )  
{  
    // conditional instruction(s) go here  
}
```


Putting it all together

Assuming a child is someone less than 13 years of age, we can re-write the initial set of instructions as follows:

```
System.out.println("How old are you?");  
age = input.nextInt();  
if (age < 13)  
{  
    System.out.print("Hello Junior!");  
}  
System.out.println("Enjoy your trip");
```

Interfacing with new code

How old are you?

10

Hello Junior!

Enjoy your trip

**first a child
approaches**

How old are you?

45

Enjoy your trip

**the second time
an adult
approaches**

IF single selection statement

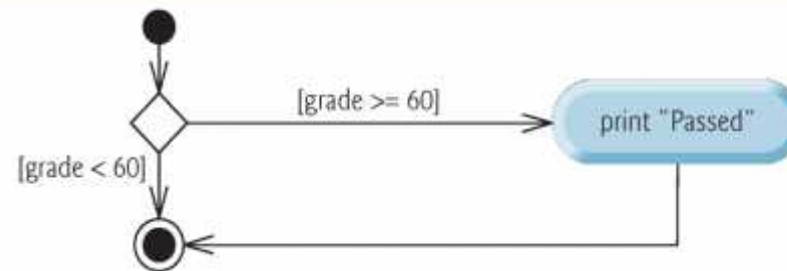
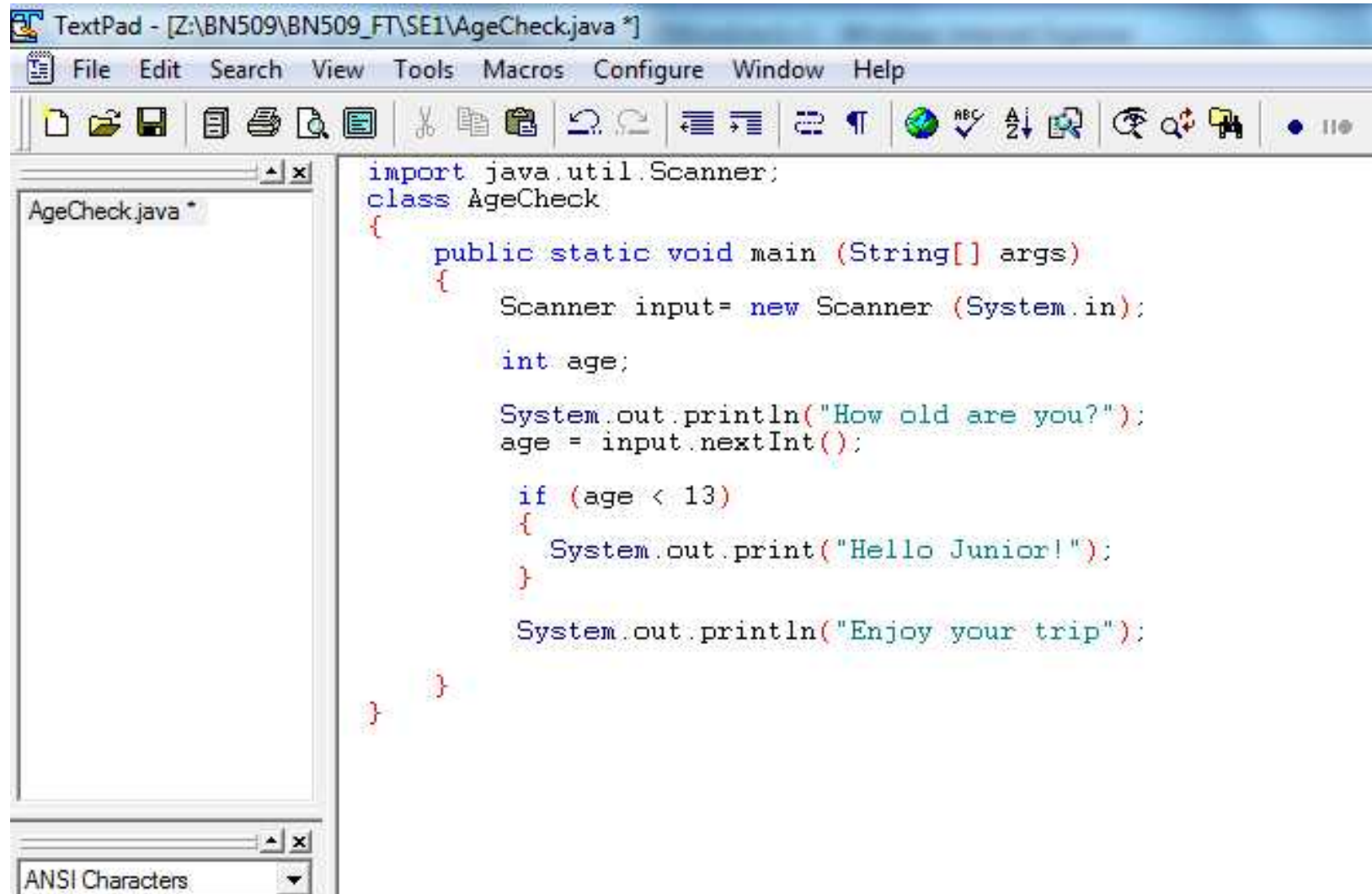


Fig. 3.2 | if single-selection statement UML activity diagram.

Program Example



The image shows a screenshot of a TextPad application window. The title bar reads "TextPad - [Z:\BN509\BN509_FT\SE1\AgeCheck.java *]". The menu bar includes "File", "Edit", "Search", "View", "Tools", "Macros", "Configure", "Window", and "Help". The toolbar contains various icons for file operations, editing, and formatting. The main text area displays the following Java code:

```
import java.util.Scanner;
class AgeCheck
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int age;

        System.out.println("How old are you?");
        age = input.nextInt();

        if (age < 13)
        {
            System.out.print("Hello Junior!");
        }

        System.out.println("Enjoy your trip");
    }
}
```

On the left side of the window, there is a file explorer pane showing "AgeCheck.java *". At the bottom of the window, there is a status bar with a dropdown menu currently set to "ANSI Characters".

The 'if...else' statement

The **if...else** statement can be used to state two alternative courses of action.

```
if ( /* test goes here */ )
{
    //instruction(s) if test is true
}
else
{
    //instruction(s) if test is false
}
```

The 'if...else' double selection statement

```
int mark;  
System.out.println("What exam mark did you get?");  
Mark = input.nextInt();  
if (mark > 39)  
{  
    System.out.println("Congratulations, you passed");  
}  
else  
{  
    System.out.println("I'm sorry, but you failed");  
}  
System.out.println("Good luck with your other exams");
```

The 'if...else' double selection statement

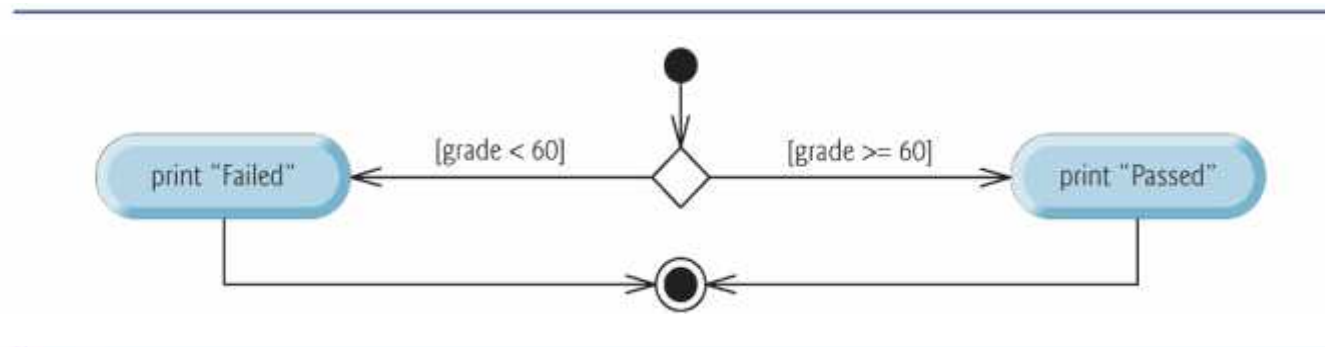
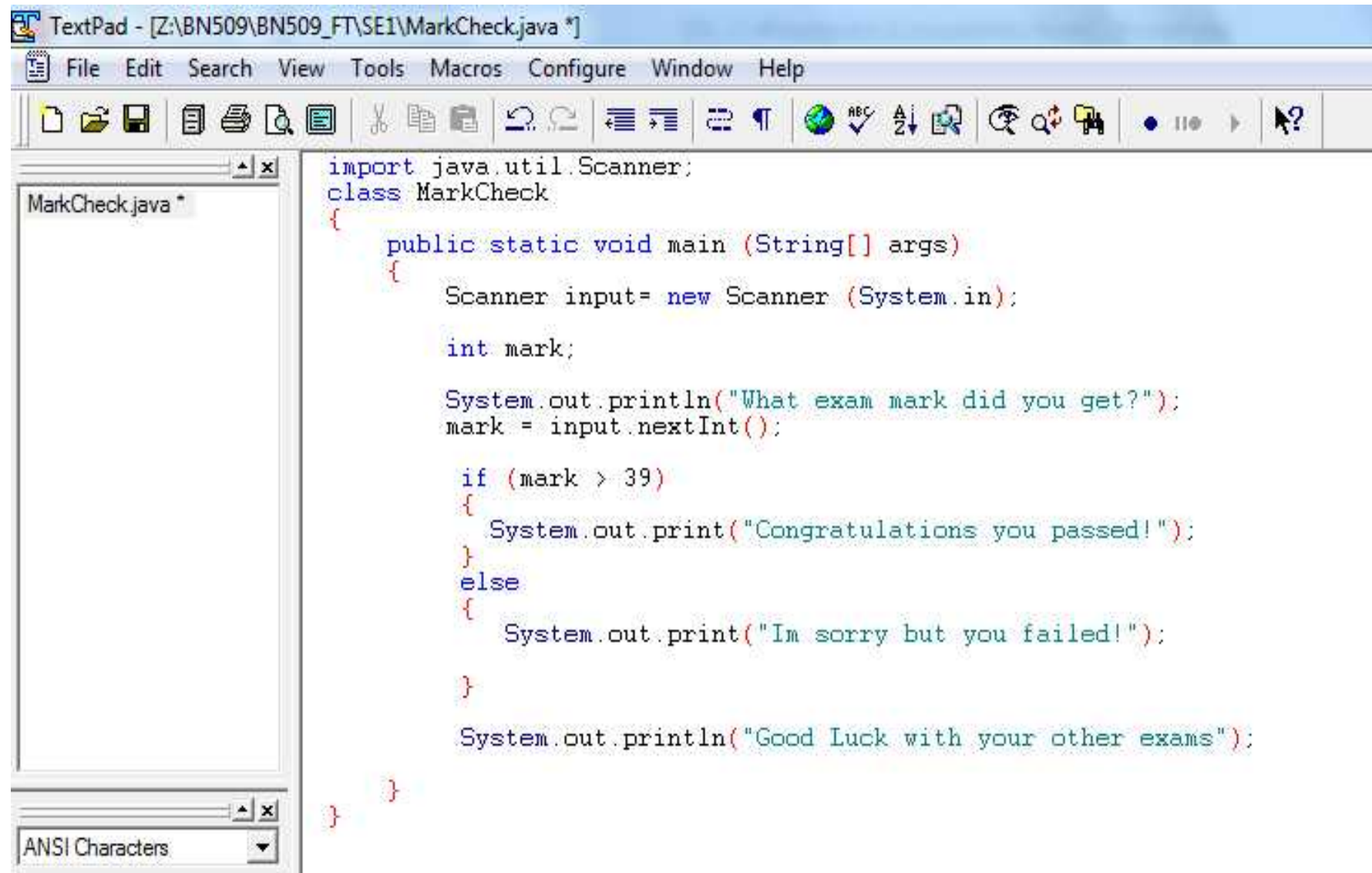


Fig. 3.3 | if...else double-selection statement UML activity diagram.

Program Example



```
import java.util.Scanner;
class MarkCheck
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int mark;

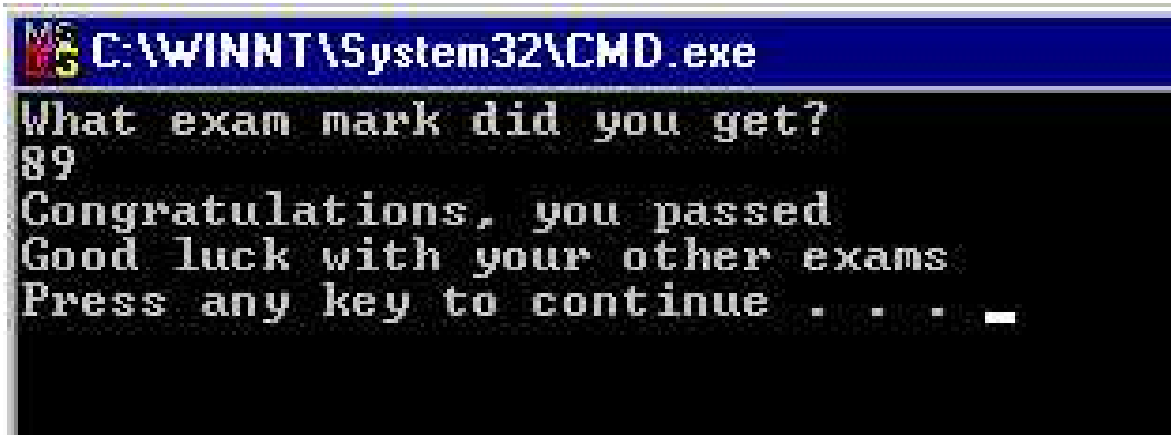
        System.out.println("What exam mark did you get?");
        mark = input.nextInt();

        if (mark > 39)
        {
            System.out.print("Congratulations you passed!");
        }
        else
        {
            System.out.print("Im sorry but you failed!");
        }

        System.out.println("Good Luck with your other exams");
    }
}
```

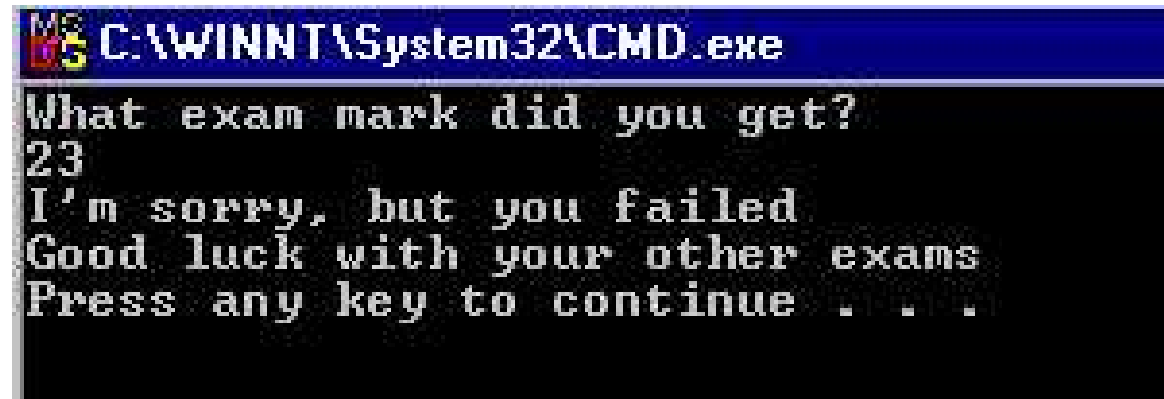

Program Example

Result if mark entered = 89



```
MS-DOS C:\WINNT\System32\CMD.exe
What exam mark did you get?
89
Congratulations, you passed
Good luck with your other exams
Press any key to continue . . .
```

Result if mark entered = 23



```
MS-DOS C:\WINNT\System32\CMD.exe
What exam mark did you get?
23
I'm sorry, but you failed
Good luck with your other exams
Press any key to continue . . .
```

Two sample program interactions

What exam mark did you get?

52

Congratulations, you passed

Good luck with your other exams

**first test run when
'if' branch is true**

What exam mark did you get?

35

I'm sorry, but you failed

Good luck with your other exams

**second test run when
'else' branch is true**

A note on the Curly Brackets

If there is **more** than **one statement** that is part of the if or else section, you need to use curly brackets.

Example

```
import java.util.Scanner;
class MarkCheck1
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

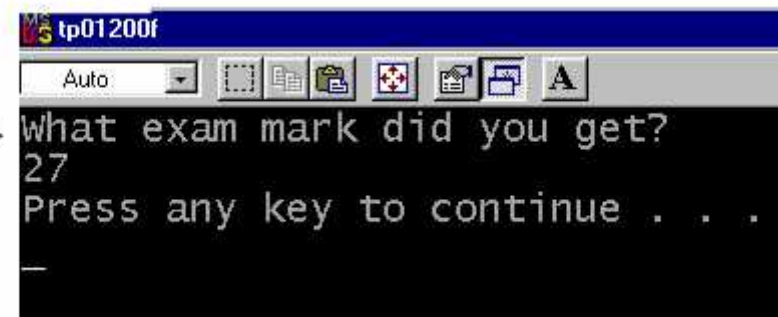
        int mark;

        System.out.println("What exam mark did you get?");
        mark = input.nextInt();

        if (mark > 39)
        {
            System.out.print("Congratulations you passed!");
            System.out.println("Good Luck with your other exams");
        }
    }
}
```

Both lines printed if mark
> 39

Result if mark
< 39



```
tp01200f
Auto
What exam mark did you get?
27
Press any key to continue . . .
_
```

A note on the Curly Brackets

Example

```
E1\MarkCheck2.java *)
ols  Macros  Configure  Window  Help

ort java.util.Scanner;
ss MarkCheck1

public static void main (String[] args)
{
    Scanner input= new Scanner (System.in);

    int mark;

    System.out.println("What exam mark did you get?");
    mark = input.nextInt();

    if (mark > 39)
    {
        System.out.print("Congratulations you passed!");
    }
}
```

**This line printed
regardless of mark**

Result →

```
MS tp0123f3
Auto
What exam mark did you get?
27
Good luck with your other exams
Press any key to continue . . .
—
```

Dangling Else Problem

It is important to note that Java always associates an **else** with the **previous if** unless told otherwise by use of the **{ }** braces.

```
if ( x > 5 )  
    if ( y > 5 )  
        System.out.println( "x and y are > 5" );  
else  
    System.out.println( "x <= 5" );
```

VS.

```
if ( x > 5 ) {  
    if ( y > 5 )  
        System.out.println( "x and y are > 5" );  
}  
else  
    System.out.println( "x <= 5" );
```

Comparison operators: an example



The following code checks whether or not an angle is a right angle:

```
if (angle == 90)
{
    System.out.println("This is a right angle");
}
```

Also known as Equality & Relational Operators



Standard algebraic
equality operator or
relational operator

Java equality
or relational
operator

Example
of Java
condition

Meaning of
Java condition

Equality operators

=

==

x* == *y

x* is equal to *y

≠

!=

x* != *y

x* is not equal to *y

Relational operators

>

>

x* > *y

x* is greater than *y

<

<

x* < *y

x* is less than *y

≥

>=

x* >= *y

x* is greater than or equal to *y

≤

<=

x* <= *y

x* is less than or equal to *y

Switch Statement

The 'switch' statement (Selection)

A **switch** statement may be used when

- only one variable is being checked in each condition
- the check involves specific values of that variable (e.g. 'A', 'B') and not ranges (e.g. >39) ;

For Example

The class is divided up into 3 groups - 1, 2, 3

If you are in group 1 your lab time is at 10.00am

If you are in group 2 your lab time is at 1.00pm

If you are in group 3 your lab time is at 11.00am

Write a program that asks the user to **input the group number** and print the **correct lab time** to the **screen**.

Solution using the If Statement

```
import java.util.Scanner;
class GroupLab
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int group;

        System.out.println("Enter your group number 1,2 or 3");
        group = input.nextInt();

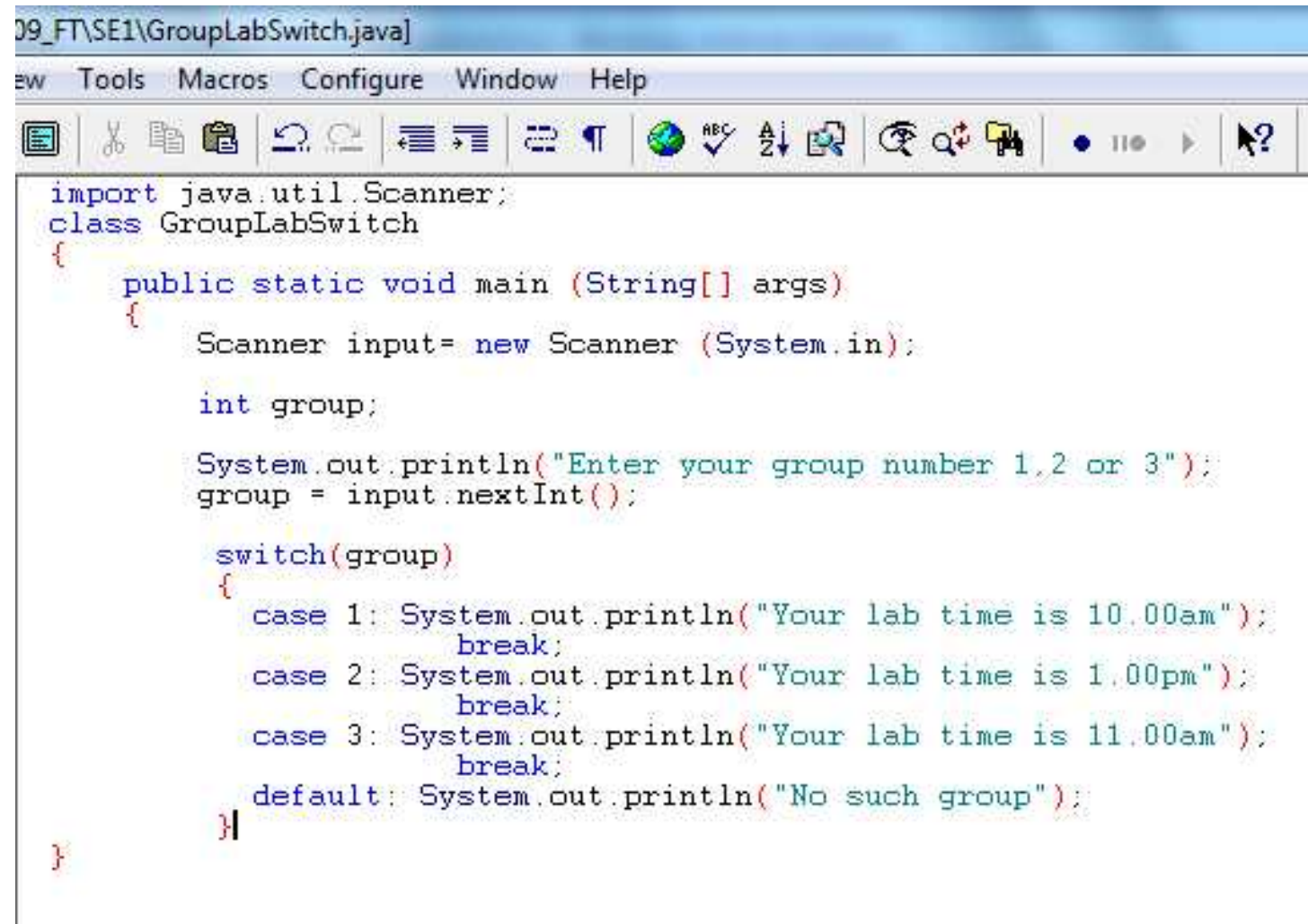
        if (group == 1)
        {
            System.out.println("Your lab time is 10.00am");
        }

        else if (group == 2)
        {
            System.out.println("Your lab time is 1.00pm");
        }

        else if (group == 3)
        {
            System.out.println("Your lab time is 1.00pm");
        }

        else
            System.out.println("Enter your group number 1,2 or 3");
    }
}
```

Solution using the Switch Statement



The screenshot shows a Java IDE window titled "09_FT\SE1\GroupLabSwitch.java". The menu bar includes "File", "Tools", "Macros", "Configure", "Window", and "Help". The toolbar contains various icons for file operations, editing, and running. The code editor displays the following Java code:

```
import java.util.Scanner;
class GroupLabSwitch
{
    public static void main (String[] args)
    {
        Scanner input= new Scanner (System.in);

        int group;

        System.out.println("Enter your group number 1,2 or 3");
        group = input.nextInt();

        switch(group)
        {
            case 1: System.out.println("Your lab time is 10.00am");
                    break;
            case 2: System.out.println("Your lab time is 1.00pm");
                    break;
            case 3: System.out.println("Your lab time is 11.00am");
                    break;
            default: System.out.println("No such group");
        }
    }
}
```

Switch Structure

The switch structure contains a series of case labels, and an optional default case.

The **break** statement causes the program control to proceed with the first statement after the **switch** structure.

The **break** statement is used because otherwise the cases in a switch statement would “*all run together*”.

If **break** is not used in a switch structure, then each time a case match occurs all the following case will be executed.

The 'switch' statement: a 2nd example

```
char group;  
System.out.println("Enter your group (A,B,C)");  
group = input.next().charAt(0);  
switch(group)  
{  
    case 'A': System.out.print("10.00 a.m ");  
                break;  
    case 'B': System.out.print("1.00 p.m ");  
                break;  
    case 'C': System.out.print("11.00 a.m ");  
                break;  
    default: System.out.print("No such group");  
}
```

Next Session

A note on Nested If Statements	
Abbreviated Assignment Expressions	
Increment and Decrement Operators	
Repetition Control Structure	
For, While, do While	
Common Escape Sequences	
Primitive Data Types	