Review

Basic Control Flow

Goal

- Control structures
- Implementing selection structure using various forms of if statements
- Learning to compare integers, floating-point numbers, chars and strings
- Understanding nested program blocks
- Understanding conditional operators
- Debugging your code

1. Simple `if` Statements

- The simplest form of a decision making structures is an **`if`** statement.
- The **`if`** statement implements a decision: <u>when a condition is fulfilled, a set of statements are executed</u>. Otherwise, the program control passes to the statements after the **`if`** block.
- The following pseudo-code shows the general format of an **`if`** statement.

```
if(condition == true)
  {
      do something;
  }
```

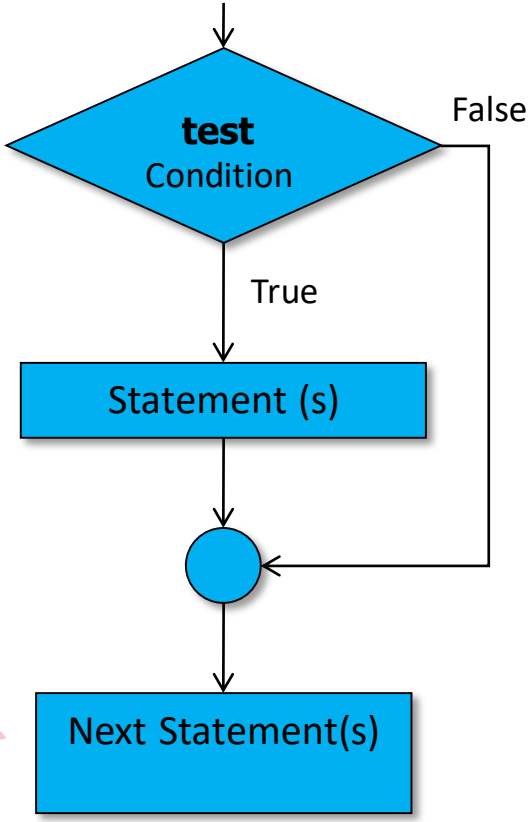
if condition

if program block

Selection Structures

```
if (test condition)
 {
  statement(s)…
 }

 Next statement(s)…
```



if Statement: Example

```
void main () {
 double price = 56.00;
 if (price > 40)
  {
   System.out.println("it's too expensive!");
     System.out.println("BYE");
  }

}
```

if condition must be boolean compatible

if block

Comparison Operators

Comparison Operators

- To compare two variables or results of expressions you can use comparison operators: **< > <= >= == !=**

Precedence	Operator	Operation name
Level 5	**<**	Comparison less-than
	<=	Comparison less-than-or-equal-to
	>	Comparison greater-than
	>=	Comparison greater-than-or-equal-to
Level 6	**==**	Comparison equal-to
	!=	Comparison not-equal-to

- For operators with double symbols don't add space between symbols
- Don't change order of symbols <= is correct but =< is **incorrect**

Comparison Operators: Examples

- A logical expression returns true (1), or false (0).

Logical expression	Meaning
quantity < 50	Compares the contents of the **quantity** variable to the number 50. The expression will evaluate to true if the **quantity** variable contains a number that is less than 50. otherwise. It will evaluate to false.
age >= 25	Compares the contents of the **age** variable to the number 25. The expression will evaluate to true if the **age** variable contains a number that is greater than or equal to 25. otherwise, it will evaluate to false.
onhand == target	Compares the contents of the **onhand** variable to the contents of the **target** variable. The expression will evaluate to true if the **onhand** variable contains a number that is equal to the number in the **target** variable; otherwise, it will evaluate to false. (Both are int variables.)
quantity != 7500	Compares the contents of the **quantity** variable to the number 7500. The expression will evaluate to true if the **quantity** variable contains a number that is not equal to 7500; otherwise, it will evaluate to false. (The quantity variable is an int variable.)

Don't check equality of real numbers

Logical Expression

- Any expression that uses any of the comparison operators: >, <, ==, >=, <=, or !=, is called a logical expression.
- A logical expression always returns true (1), or false (0). The result is **boolean**
- Example:

```
int x = 7,  y = 9;
boolean b;
b = x < y;
System.out.println("The value b is: ", b);
```

- The above code segment prints:

```
The value of b is: true
```

How to Represent Conditions?

- The test condition in an if-statement can be a constant, a variable, or an expression that can be logically interpreted as **true** or **false**.

if condition must be boolean compatible

- *Use Comparison Operators*

< >= > <= == !=

to compose a logical expression and compare numbers, char and strings.

Remember:

Only use **==** *inside* condition tests

use **=** *outside* condition tests

if Statement: Brace Layout

- Making your code easy to read is good practice.
- Lining up braces vertically helps.

```
if (price > 40)
{
    System.out.println("It's too expensive.");
    System.out.println("Bye…");
}
```

- Or

```
if (price > 40) {
    System.out.println("It's too expensive.");
    System.out.println("Bye…")
}
```

Another style

Indent when Nesting

Block-structured style: *nested* statements are indented by one or more levels.

```
void main()
{
    int floor;
    ...
    if (floor > 13)
    {
        floor--;
    }
    ...

}
```

0 1 2

Indentation level

if Statement: Variations

if Statement with one Executable Line

- If your if-block has only one statement, you have the option of eliminating the opening and closing braces:

```
void main(){
   double price = 56.00;
   if (price > 40) {
      System.out.println("It's too expensive.");
   }
}
```

If block with one executable line

It is a good programming practice to keep the braces. It makes code easier to read and debug.

Common Errors

```
if (price > 40);
{
   System.out.println("It's too expensive.");
}
```

How does this code execute? Throws a compiler error?

No!

The compiler does not complain.

It interprets this `if` statement as follows:

If price is greater than 40, execute the do-nothing statement. (semicolon by itself is the do nothing statement)

Then after that execute the code enclosed in the braces. Any statements enclosed in the braces are no longer a part of the `if` statement.

Can I use = instead of ==

- NO!

```
void main()
{
    int x = 0;
    if(x = 5)
        System.out.println(x);
}
```

Will cause compilation error!

- The output is: The value of x is: 5

Condition: `char` Type

- Char data type can be used inside a condition test

```
void main(){
   char answer = 'Y';
   if (answer == 'Y')
      System.out.println("Answer is yes.");
}
```

This condition will be evaluated as true

- The output is: Answer is yes.

Condition: `string` Type

- We can compare strings using == or the .equals() method

```
void main(){
  String fruit = "Apple";
  if (fruit == "Peach")
     System.out.println("Entered the if");
}
```

What about this one?

```
void main(){
  String fruit = "Apple";
  if (fruit.equals("Peach"))
     System.out.println("Entered the if");
}
```

Selection structures

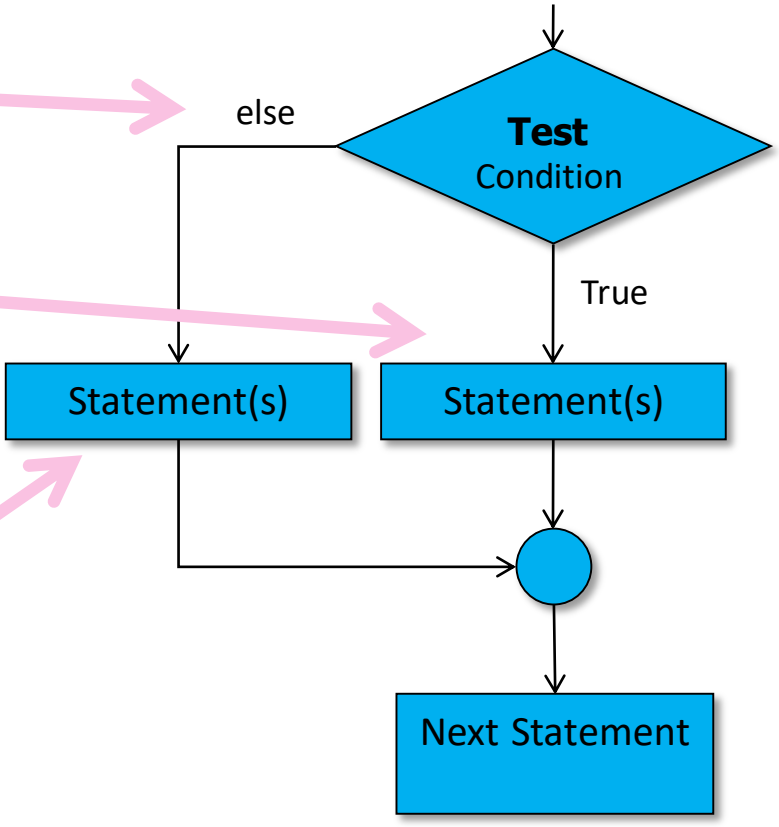
`if … else`

statement

2. `if…else` Statement

To design a two way decision structure

```
if (test condition)
{
   statement(s)…
}
else
{
   statement(s)…
}
```



The `if…else` Statement

If it happens that there is nothing
to do in the **`else`** branch of the statement.

So don't write it.

It will become the simple if statement

The `if…else` Statement

- Again you can eliminate braces if you have only one executable statements

```
void main () {
      double price = 56.00;
      if (price > 40 )
         System.out.println("It's too expensive.");
      else
         System.out.println("Let's buy it.");
}
```

if block with one executable line

else block with one executable line

Removing Duplication

```
if (floor > 13)
{
   actual_floor = floor - 1;
   System.out.println("Actual floor: ",
   actual_floor);
}
else
{
   actual_floor = floor;
   System.out.println("Actual floor: ",
   actual_floor);
}
```

Always ask yourself:
Do these statements depend on the test?

Removing Duplication

```
if (floor > 13)
{
   actual_floor = floor - 1;
}
else
{
   actual_floor = floor;
}
System.out.println ("Actual floor: ", actual_floor);
```

You should remove this duplication

Selection structures

Multi-way

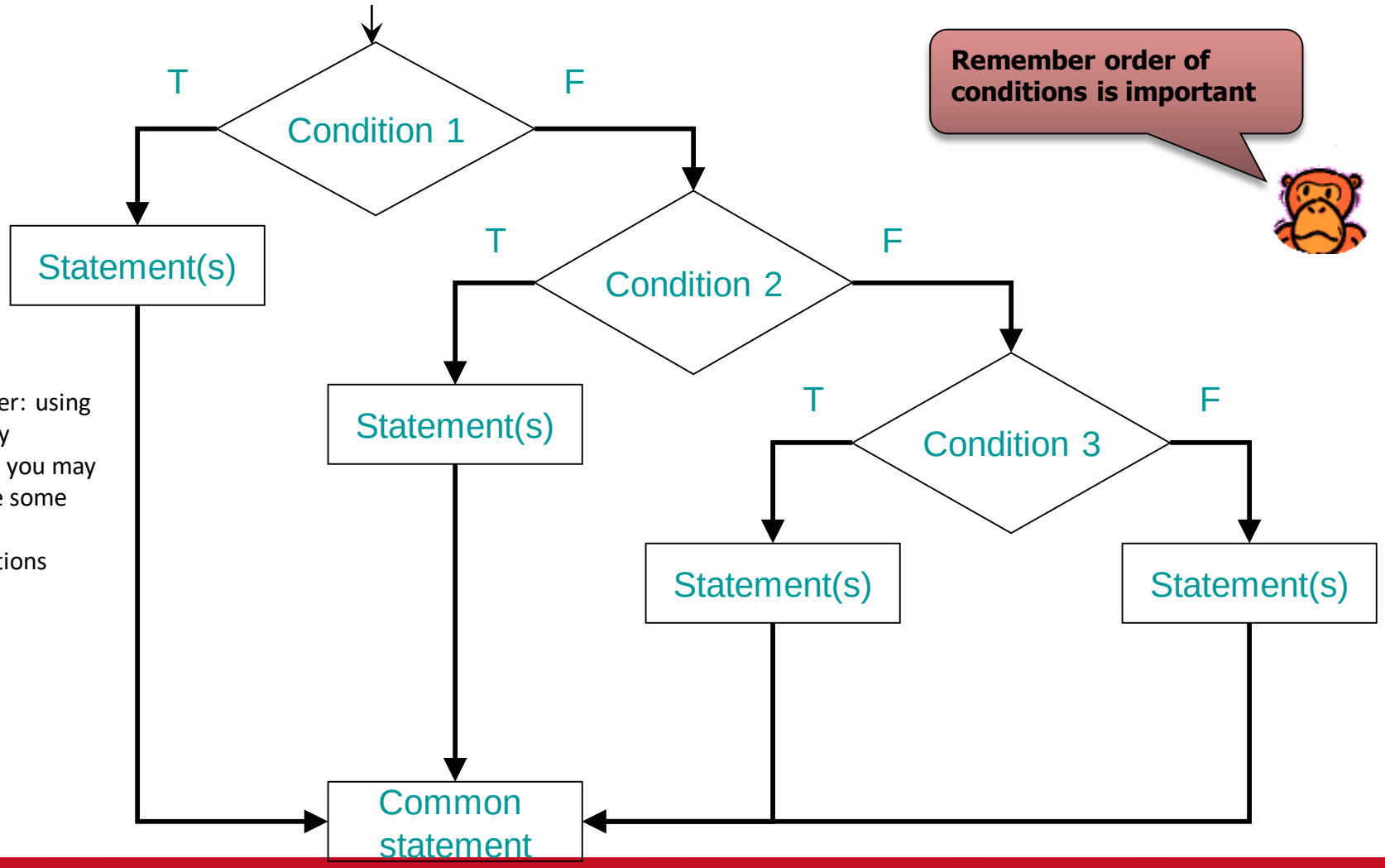
Selection Structures

Multi-way Selection Structures

- Multiple if statements can be combined to implement complex decisions

- Two way to implement a multi-way decision:
  - Using **`if… else if… else if… else`**
  - Using **`switch`** statement – Not Covered in ENGG 233

3. Multi-Way Selections

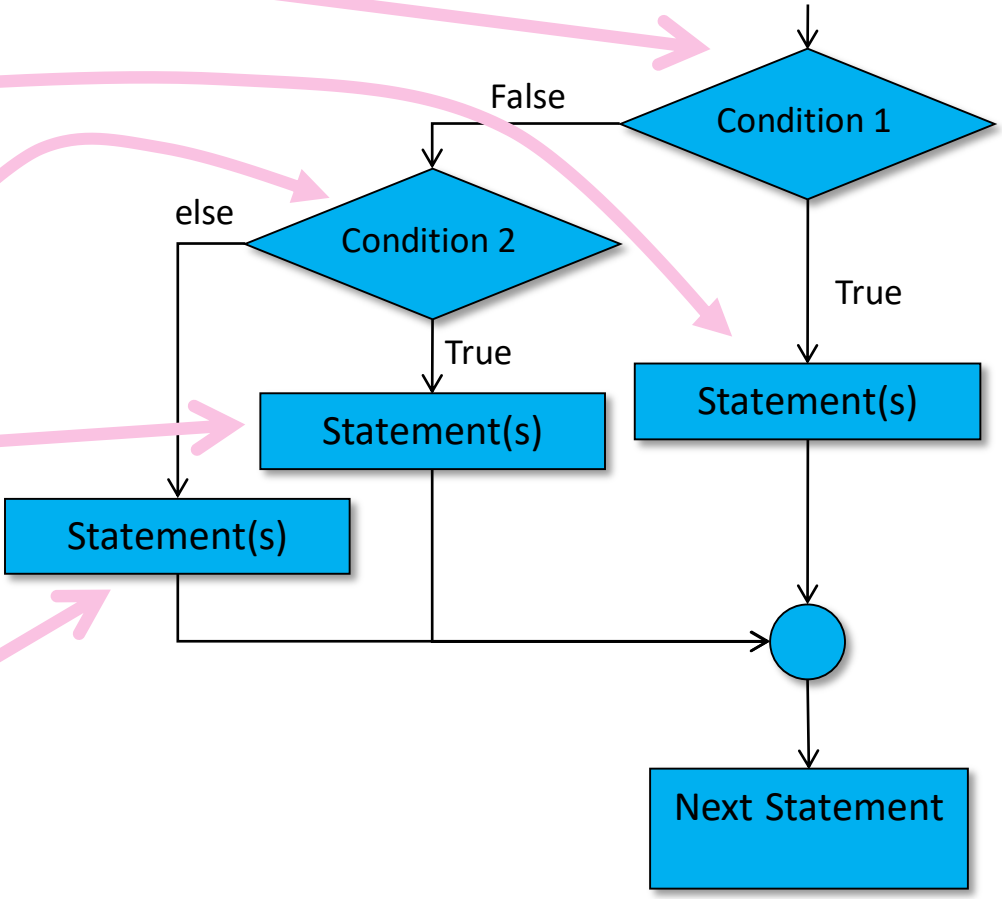


Remember order of conditions is important

Remember: using multi-way selection you may eliminate some possible combinations

if … else if … else Statement

```
if(condition1)
{
  statement(s)…
}
else if (condition2)
{
  statement(s)…
}
else
{
  statement(s)…
}
```



if … else if … else Statement

```
void main (){
     double price = 56.00;
     if (price > 40 )
     {
        System.out.println("It's too expensive. ");
        System.out.println("Bye…");
     }
     else if (price > 30)
    {
       System.out.println("It's a good deal! ");
       System.out.println("I may but it. ");
    }
    else
    {
     System.out.println("It's a great deal! ");
     System.out.println("I will buy it now. ");
    }
 }
```

You can have several else **if** blocks

Another Example

```
void main (){
  double price = 56.00;

  if (price > 40)
      System.out.println("It's too expensive.
  ");
  else if (price > 35)
      System.out.println("It's still expensive.
  ");
  else if (price > 30)
      System.out.println("I may but it. ");
  else {
     System.out.println("It's a great deal! ");
     System.out.println("I will buy it now. ");
    }
}
```

Notice: braces are removed

Logical Operators

Logical Operators

- Combine two or more conditions into one compound condition
- The following operator are called logical operators:

Name	Operator	Description
And	&&	Returns true if both the left operand and right operand return a value of true; otherwise, it returns a value of false
Or	||	Returns true if either the left operand or right operand returns a value of true; if neither operand returns a value of true, then the expression containing the Or || operator returns a value of false
Not	!	Returns true if an expression is false and returns false if an expression is true

Not Operator (!)

- **Example 1:**

```
boolean a = false;

if (!a)
   System.out.println("Today is a nice day");
```

- **Example 2:**

```
 int  a = 50;

if (!(a > 20))
   System.out.println("surprise!");
```

And Operator (&&)

- **Example 1: age must be greater and equal to 0 and less than and equal to 99**

```
int  age = 50;

if ((age >= 0) && (age <= 99))
   System.out.println("age is valid! ");
```

- **Example 2:**

```
int  x = 7,  y = 9,  a = 2,  b = 3 ;

if ((x < y) && (b > a))
   System.out.println(" x is less than y and a is less than b. ");
```

OR Operator (||)

- **Example: If temperature is more than 25 or less than 0, print: “The temperature is not desirable!”**

```
int  temperature = 7;

if ((temperature  < 0) || (temperature > 25))
   System.out.println("The temperature is not desirable!");
```

True & False Table

Truth table for the && (And) operator		
Value of *condition1*	Value of *condition2*	Value of *condition1 && condition2*
true	true	true
true	false	false
false	true	false
false	false	false

Truth table for the || (Or) operator		
Value of *condition1*	Value of *condition2*	Value of *condition1 || condition2*
true	true	true
true	false	true
false	true	true
false	false	false

Nested if and else statements

- **Consider:**

```
int  x = 7,  y = 9,  a = 2,  b = 3 ;

if ((x < y) && (b > a))
   System.out.println(" x is less than y and a is less than b. ");
```

- **The above can be written as follows:**

```
int  x = 7,  y = 9,  a = 2,  b = 3 ;
if (x < y)
{
    if (b > a)
    {
      System.out.println(" x is less than y and a is less than b.");
    }
}
```