

## **Computer programming (1)**



# Chapter 1

## CSC101 Review



# Agenda

- **Java Basics**
- **Conditional Statements**
  - if Statements
  - if-else Statements
  - Nested if-statements
- **Repetition Statements**
  - while
  - do-----while
  - for
- **Common Programming Errors**



# Agenda

- **The API Library**
  - Math Classes
  - The String Class
- **Array**
  - Processing Array Elements
  - Passing Arrays to Methods



+

# Java Basics



# Java Environments

- Java Development Environments

- Eclipse

- Netbeans

- Notepad and Command Line

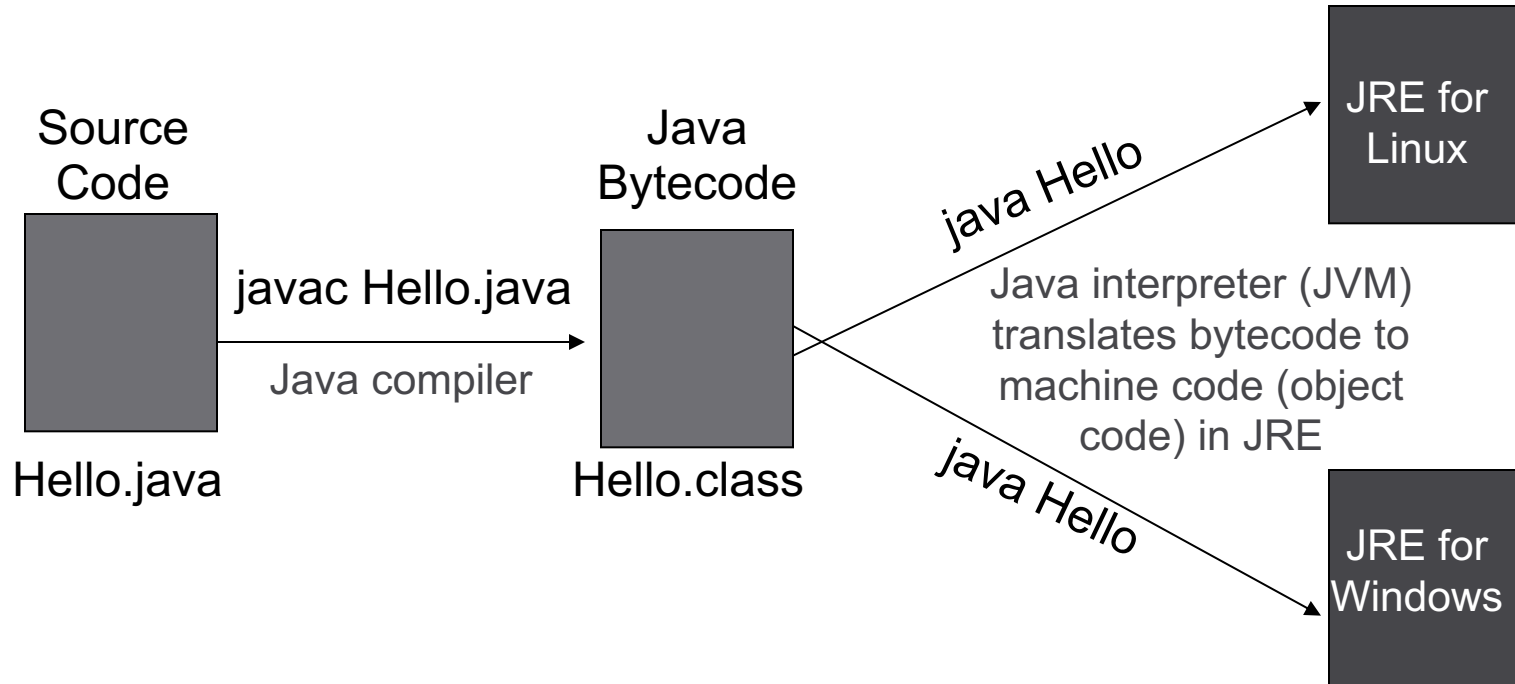
- Or you can do it online:

- [http://www.tutorialspoint.com/compile\\_java\\_online.php](http://www.tutorialspoint.com/compile_java_online.php)

+

# Running and Compiling Java

7



**JRE(Java Runtime Environment)** contains class libraries which are loaded at runtime.

# + Methods in Java

The *main* method has a specific signature.

## ■ Example: “Hello world!” Program in Java

```
public class Hello
{
    public static void main(String args[])
    {
        System.out.println(“Hello world!”);
    }
}
```

← **Notice no semi-colon at the end!**



# + Methods in Java (cont.)

- All methods must be defined inside a class.
- Format for defining a method:

```
[modifiers] return_type method_name([param_type param]*)  
{  
  
    statements;  
  
}
```



# Important Java Concepts

- Everything in Java must be inside a class.
- Every file may only contain one public class.
- The name of the file must be the name of the class appended to the java extension.
- Thus, *Hello.java* must contain one public class named *Hello*.

# Outline



**The if Statement and Conditions**

**Other Conditional Statements**

**The while Statement**



**Other Repetition Statements**

**The API Library**

**Array**

# Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next
- Therefore they are sometimes called *selection statements*
- Conditional statements give us the power to make basic decisions
- The Java conditional statements are the:
  - *if statement*
  - *if-else statement*
  - *switch statement*

# The if Statement

- An example of an if statement:

```
if (sum > MAX)
    delta = sum - MAX;
System.out.println ("The sum is " + sum);
```

- First the condition is evaluated -- the value of sum is either greater than the value of MAX, or it is not
- If the condition is true, the assignment statement is executed -- if it isn't, it is skipped.
- Either way, the call to println is executed next

- Expressions that use logical operators can form complex conditions

```
if (total < MAX+5 && !found)
    System.out.println ("Processing...");
```

- All logical operators have lower precedence than the relational operators
- Logical **NOT** has higher precedence than logical AND and logical OR

# Outline

The `if` Statement and Conditions



Other Conditional Statements

The `while` Statement



Other Repetition Statements

The **API Library**

**Array**

- Several statements can be grouped together into a *block statement* delimited by **braces**
- A block statement can be used wherever a statement is called for in the Java syntax rules

```
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
```



- In an if-else statement, the if portion, or the else portion, or both, could be block statements

```
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
else
{
    System.out.println ("Total: " + total);
    current = total*2;
}
```

# + Nested if Statements

- The statement executed as a result of an if statement or else clause could be another if statement
- These are called *nested if statements*
- An Braces can be used to specify the if statement to which an else clause belongs
- **Example :**

```
if(Boolean_expression 1){  
    // Executes when the Boolean expression 1 is true  
        if(Boolean_expression 2){  
            // Executes when the Boolean expression 2 is true  
                }  
        }  
}
```

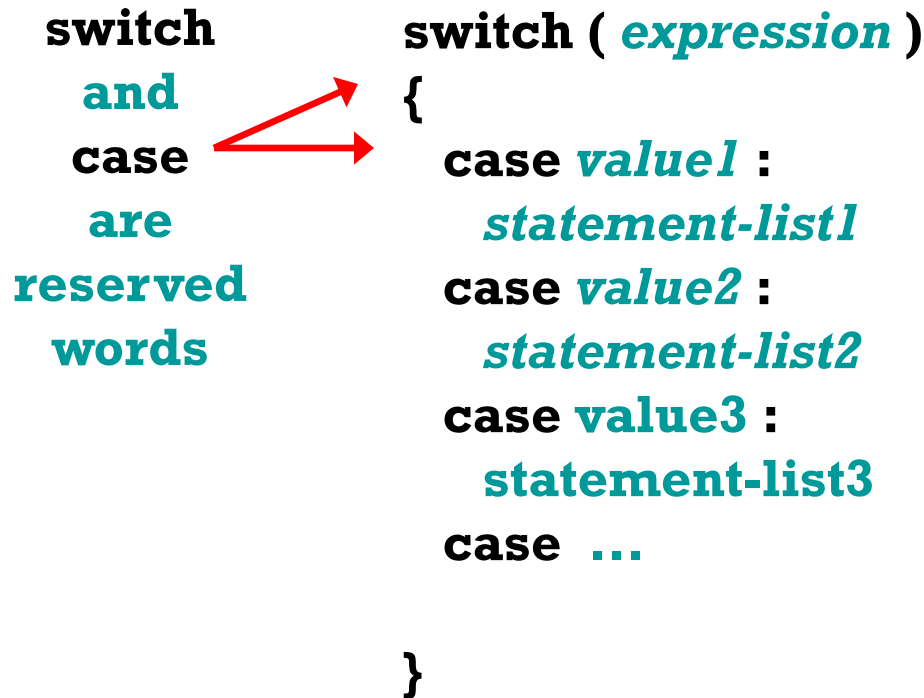
# + The switch Statement

- The *switch statement* provides another way to decide which statement to execute next
- The switch statement evaluates an expression, then attempts to match the result to one of several possible cases
- Each **case** contains a value and a list of statements
- The flow of control transfers to statement associated with the first case value that matches

# + The switch Statement

- The general syntax of a **switch** statement is:

**switch**  
**and**  
**case**  
**are**  
**reserved**  
**words**



```
switch ( expression )  
{  
  case value1 :  
    statement-list1  
  case value2 :  
    statement-list2  
  case value3 :  
    statement-list3  
  case ...  
}
```

If *expression*  
matches *value2*,  
control jumps  
to here

# + The switch Statement

- Often a *break* statement is used as the last statement in each case's statement list
- A break statement causes control to transfer to the end of the switch statement
- If a break statement is not used, the flow of control will continue into the next case
- Sometimes this may be appropriate, but often we want to execute only the statements associated with one case

# + The switch Statement

- An example of a switch statement:

```
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```

# + The switch Statement

- A switch statement can have an optional *default* case
- The default case has no associated value and simply uses the reserved word default
- If the default case is present, control will transfer to it if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch

+

```
public class Test {  
  
    public static void main(String args[]) {  
        // char grade = args[0].charAt(0);  
        char grade = 'C';  
  
        switch(grade) {  
            case 'A' :  
                System.out.println("Excellent!");  
                break;  
            case 'C' :  
                System.out.println("Well done");  
                break;  
            case 'D' :  
                System.out.println("You passed");  
                break;  
            case 'F' :  
                System.out.println("Better try again");  
                break;  
            default :  
                System.out.println("Invalid grade");  
        }  
        System.out.println("Your grade is " + grade);  
    }  
}
```



# + The switch Statement

- The expression of a switch statement must result in an *integral type*, meaning an **integer** (byte, short, int, long) or a **char**
- It cannot be a boolean value, a floating point value (float or double), or another integer type
- You cannot perform relational checks with a switch statement

# Outline

**The `if` Statement and Conditions**

**Other Conditional Statements**



**The `while` Statement**

**Other Repetition Statements**

**+**

**The API Library**

**Array**

# The while Statement

- An example of a while statement:

```
int count = 1;
while (count <= 5)
{
    System.out.println (count);
    count++;
}
```

- If the condition of a while loop is false initially, the statement is never executed
- Therefore, the body of a while loop will execute zero or more times



# Infinite Loops

- The body of a while loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally



# Infinite Loops

- An example of an infinite loop:

```
int count = 1;  
while (count <= 25)  
{  
    System.out.println (count);  
    count = count - 1;  
}
```



# Nested Loops

- Similar to nested if statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely



# Nested Loops

- How many times will the string "Here" be printed?

```
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 <= 20)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

**10 \* 20 = 200**

# Outline

The `if` Statement and Conditions

Other Conditional Statements

The `while` Statement

 Other Repetition Statements

+

The API Library

Array



# The do Statement

- An example of a do loop:

```
int count = 0;  
do  
{  
    count++;  
    System.out.println (count);  
} while (count < 5);
```

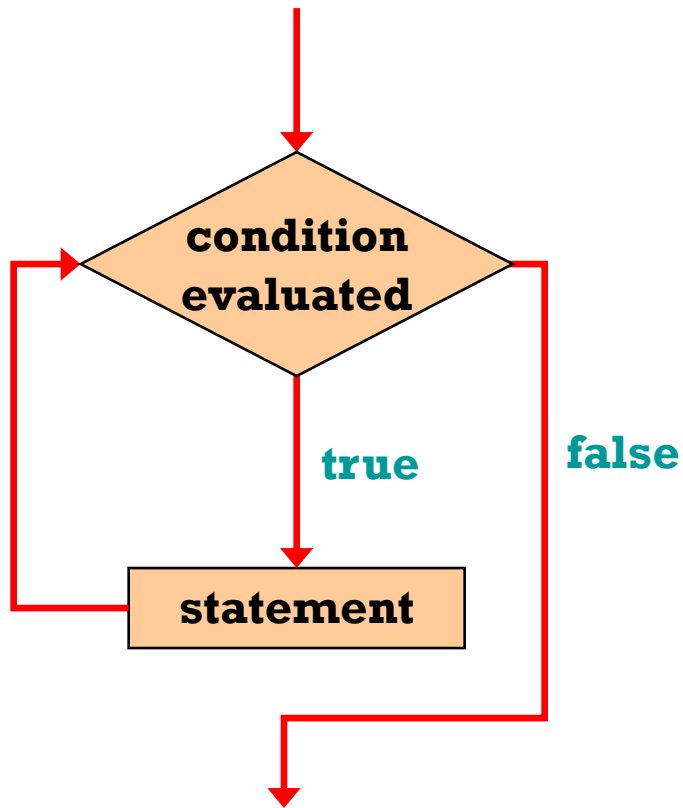
- The body of a do loop executes **at least once**

+

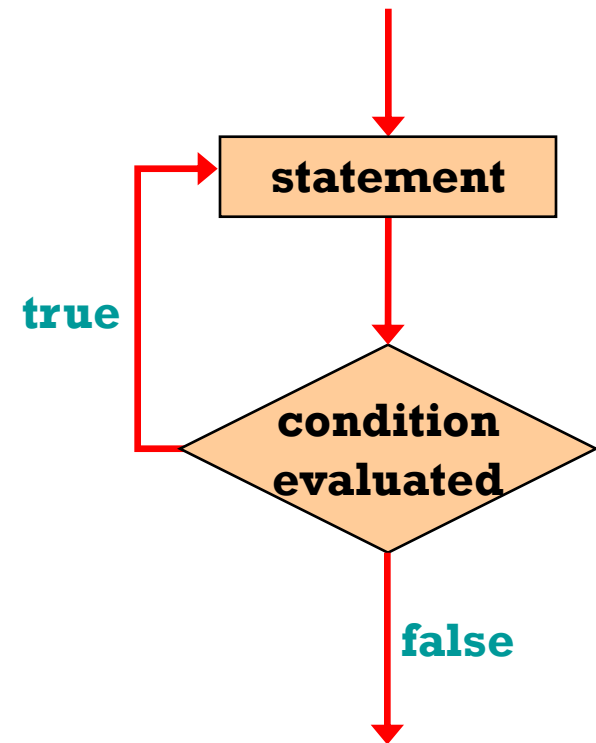
# Comparing while and do

34

## The while Loop



## The do Loop



# The for Statement

- An example of a for loop:

```
for (int count=1; count <= 5; count++)  
    System.out.println (count);
```

- The initialization section can be used to declare a variable
- Like a while loop, the condition of a for loop is tested prior to executing the loop body
- Therefore, the body of a for loop will execute zero or more times

# The for Statement

- The increment section can perform any calculation

```
for (int num=100; num > 0; num -= 5)  
    System.out.println (num);
```

- A for loop is well suited for executing statements a specific number of times that can be calculated or determined in advance

# + Loop Comparison

for loop:

## When to use

If you know, prior to the start of loop, how many times you want to repeat the loop.

## Template

```
for (int i=0; i<max; i++)
{
    <statement(s)>
}
```

do loop:

If you always need to do the repeated thing at least one time.

```
do
{
    <statement(s)>
    <prompt - do it again (y/n)?>
} while (<response == 'y'>);
```

while loop:

If you can't use a for loop or a do loop.

```
<prompt - do it (y/n)?>
while (<response == 'y'>)
{
    <statement(s)>
    <prompt - do it again (y/n)?>
}
```

# + Programming Errors

- **Compilation errors**
  - Syntax error (example: missing a semi-colon).
  - Semantic error. (For example, applying modulus % on floating-point value for certain programming languages. In Java ,is it fine? Yes!)
  - Easiest type of errors to fix.
- **Runtime errors**
  - Occur at runtime.
  - Java's exception mechanism can catch such errors.
- **Logic errors**
  - Program runs but produces incorrect result.
  - Hard to characterize, hence hardest to fix.
- **Programming errors are also known as bugs**



# Common Programming Errors (if Statement)

■ `if (0 <= x <= 4)` → → → `if (0<=x && x<=4)`

■ `=` → → → `==`



# Common Programming Errors (switch Statement)

- Make sure the controlling expression and case labels are of the same permitted type (int or char or string)
- Include a default case
- Enclosed in one set of braces
- Each alternative is ended by a break statement



# Outline

**The if Statement and Conditions**

**Other Conditional Statements**

**The while Statement**

**+**

**Other Repetition Statements**



**The API Library**

**Array**

- It is legal to pass an integer value to a method that accepts a floating-point argument. Note the following example. Horton's Law says that the length of a river is related to the area drained by the river in accordance with this formula:

$$\text{length} \approx 1.4 (\text{area})^{0.6}$$

- Here's how to implement Horton's Law in Java code:

```
int area = 10000; // square miles drained  
double Length = 1.4 * Math.pow(area, 0.6);
```

OK to pass an  
`int (area)`,  
into `pow`,  
which accepts  
double  
arguments.



# String Class

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4



# Quick Check

**What output is produced by the following?**

```
String str = "Space, the final frontier.";
System.out.println (str.length());
System.out.println (str.substring(7));
System.out.println (str.toUpperCase());
System.out.println (str.length());
```

```
26
the final frontier.
SPACE, THE FINAL FRONTIER.
26
```

# Outline

**The if Statement and Conditions**

**Other Conditional Statements**

**The while Statement**

**+ Other Repetition Statements**

**The API Library**

 **Array**

## + Processing Array Elements

### Copying The Contents of One Array to Another Array

- To copy the contents of one array to another you must copy the individual array elements.
- To copy the contents of the array referenced by *oldValues* to the array referenced by *newValues* we could write:

## + Processing Array Elements

### Copying The Contents of One Array to Another Array

47

```
int count;
short[ ] oldValues = {10, 100, 200, 300};
short[ ] newValues = new short[4];
    System.out.println("newValues before copying values from
oldValues ");
        for (int value : newValues)
            System.out.println(value);
// If newValues is large enough to hold the values in oldValues
if (newValues.length >= oldValues.length)
{
    for (count = 0; count < oldValues.length; count++)
    {
        newValues[count] = oldValues[count];
    }
}
    System.out.println("newValues after copying values from
                                oldValues");
        for (int value : newValues)
            System.out.println(value);
```

## + Processing Array Elements

### Copying The Contents of One Array to Another Array

```
newValues before copying values from oldValues
0
0
0
0
newValues after copying values from oldValues
10
100
200
300
```





## Passing an Array as an Argument to a method

- Write a java class called “ExampleArrays.java” that represents the following instance data:
  - One array “Arr=11,30,32,43,23,45,6,76” and one variable “key” which value is 43.
  - Method search() searches the required value “Key” in an array “Arr” and return the index.
  - Method avrage() to calculate and print the average of the elements of an array.

## The program:

```
public class ExampleArrays{
    public static void main(String[] args)
    {
        int Arr[]={11,30,32,43,23,45,6,76};
        int key = 43;
        int result1 = Search(Arr ,key);
        Average(Arr);
        System.out.println("the required value is at index = " + result1);
    }
}
```

```
public static int Search(int[] A, int N)
{
    int index = 0;
    for (int i = 0; i < A.length ;i++)
    {
        if ( A[i] == N )
            index = i; // N has been found at this index!
    }
    return index;
}
```



## The program:

```
public static void Average(int[] A)
{
    int sum = 0;
        for(int i=0; i < A.length ; i++)
            sum = sum + A[i];
        //calculate average value
    double average = (sum*1.0) / A.length;
    System.out.println("Average value of array elements is : " + average);
}
}
```



+

**Thanks!**