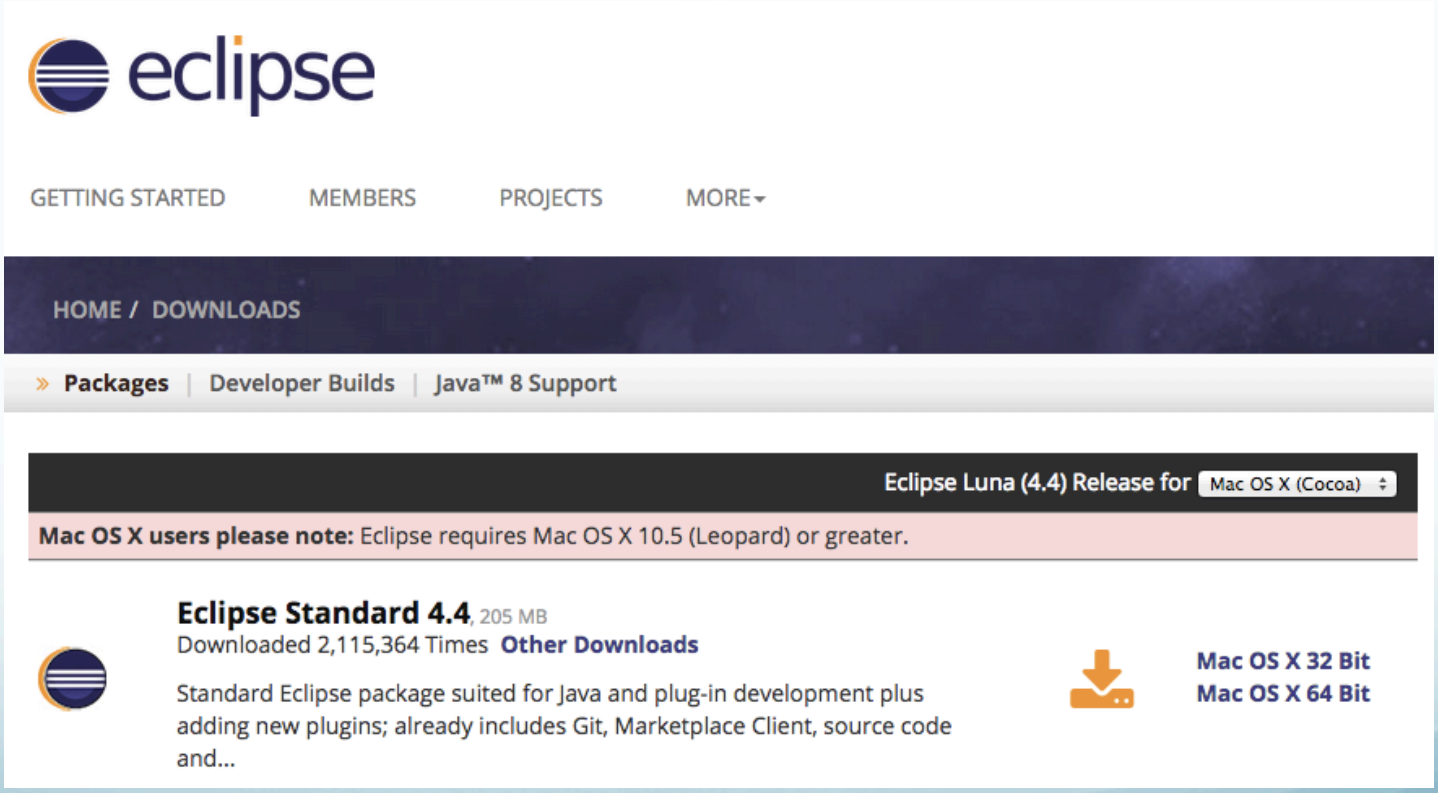
Java Basics I

COMP3111/H Tutorial1

Eclipse (IDE)

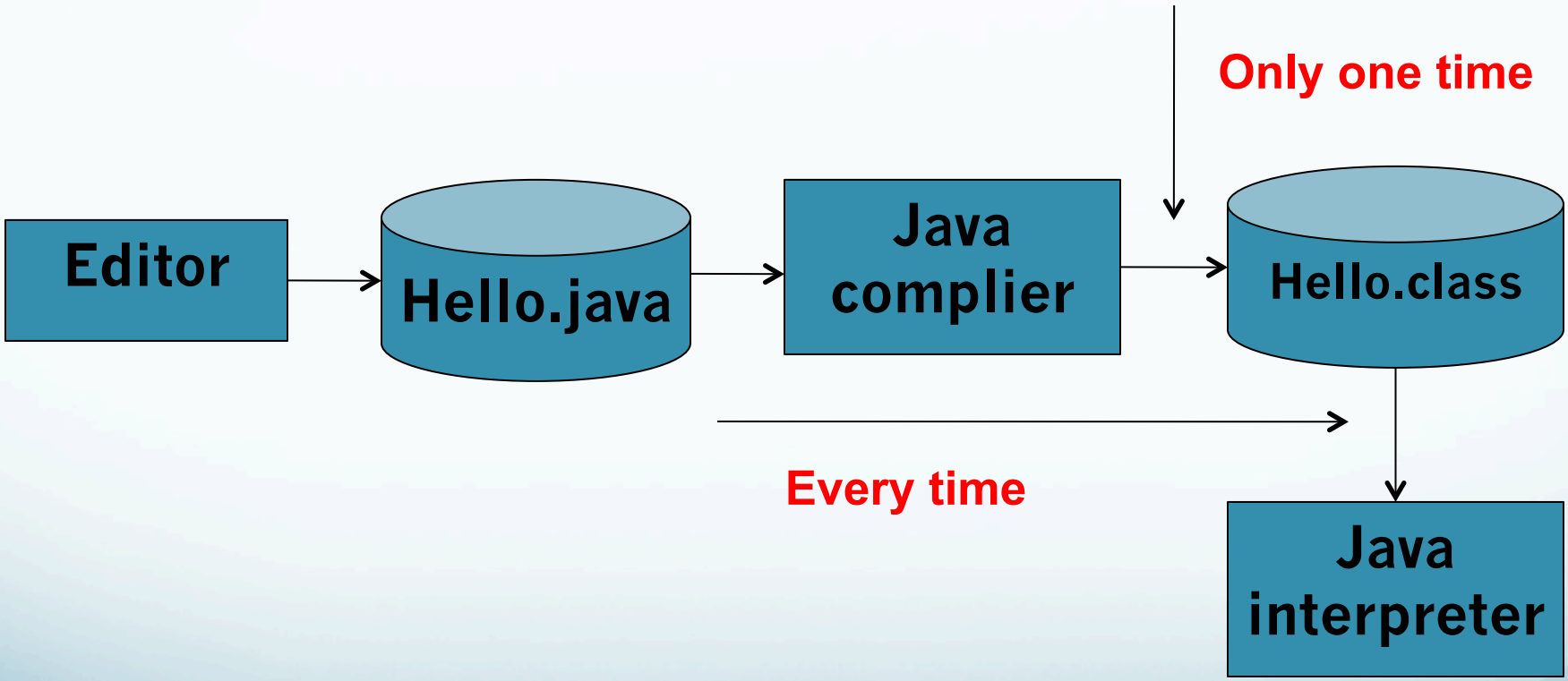
- Cross-platform (Windows / Mac OS X)



Eclipse(IDE)

- How to create a simple HelloWorld project?
- How to import/export an existing project?
  - It is useful when you need to share an initial codebase to other developers
  - However, once you learned how to use a version control system (e.g. CVS, SVN, GIT...), you seldom need to import/export projects

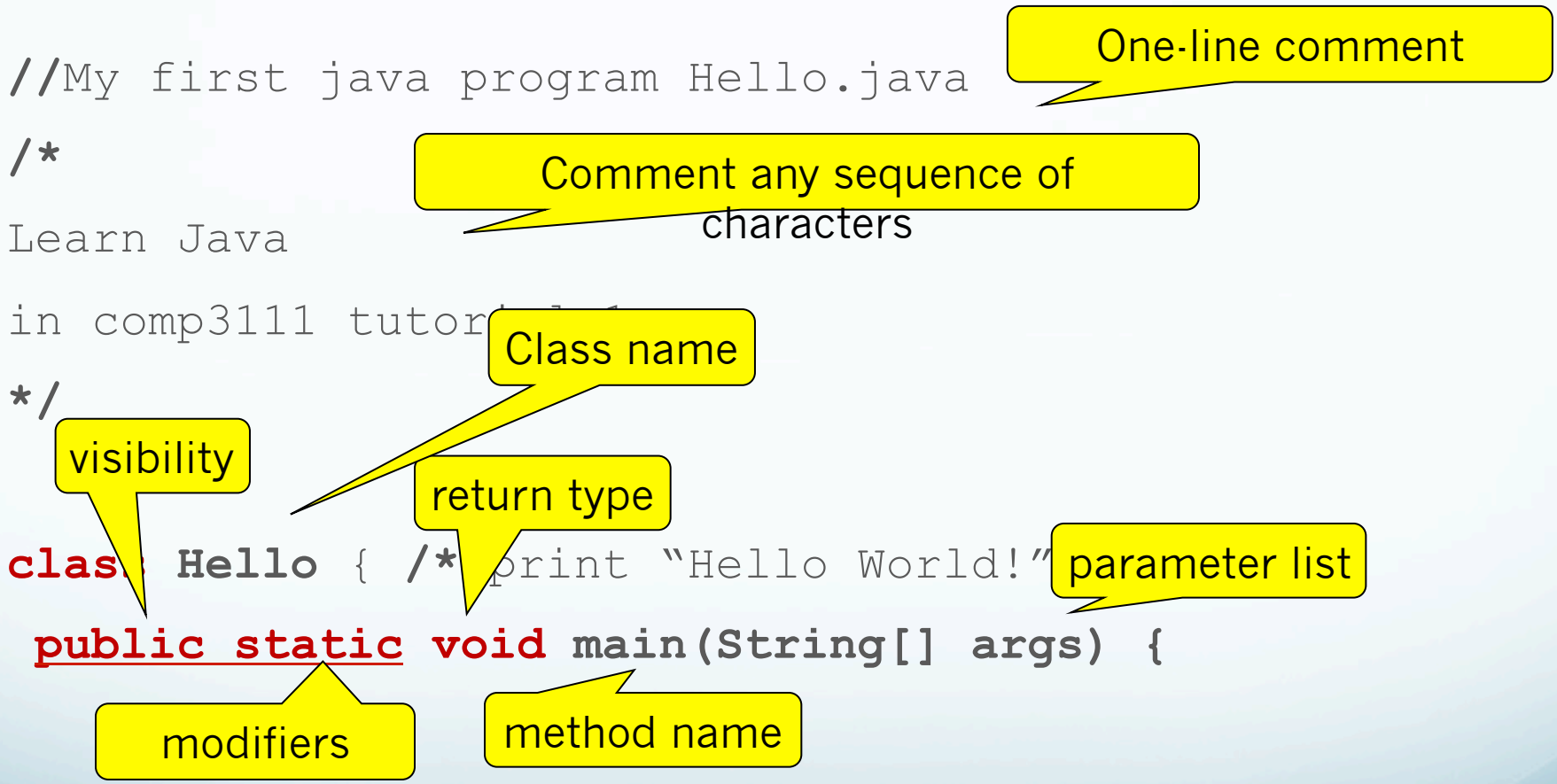
Phases of a Java program



HelloWorld in Java

```
//My first java program
/*
Learn in comp3111 tutorial 1
*/
class Hello { /* print “Hello World!” */
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

HelloWorld in Java



Java Basics (Part I)

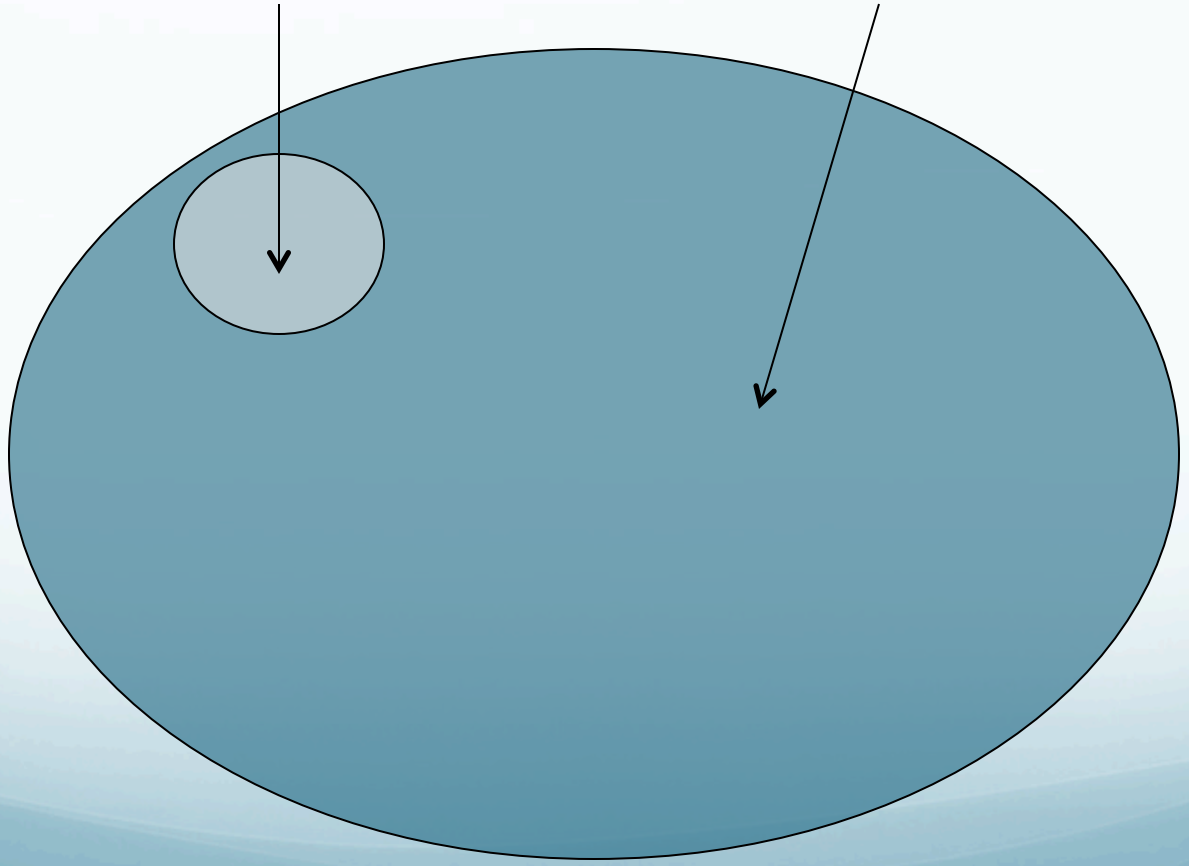
- Types and Variables
- Expression
- Branching
- Loops
- String
- Array

Identifiers

- In Java, An identifier is a name given to a package, class, interface, method, or variable
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($)
- An identifier must start with a letter, an underscore (_), or a dollar sign ($)
- It cannot start with a digit
- It cannot be a reserved word such as *true*, *false*, *null*, *if*….

Java Types

- There are two kinds of *types* in the Java programming language: **primitive types** and **reference types**



Primitive types

Name	Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127)	8-bit signed
short	-2^{15} (-32768) to $2^{15}-1$ (32767)	16-bit signed
int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
long	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Declaring variables of primitive types

- **byte**
  - byte b = 10;
- **short**
  - short s = 10;
- **int**
  - int i = 1;
- **long**
  - long l = 100;
- **float**
  - float f = 10.4f;
- **double**
  - double d = 10.10;
- **boolean**
  - boolean flag = true
- **char**
  - char ch = 'a';

Primitive types: Java V.S. C++

- Except some minor differences (e.g. “bool” in C++ becomes “boolean” in Java), declaring variables of primitive types is quite similar in C++ and Java
- Java has a stronger type-checking to avoid programmers to make mistakes. For example:
  - float f = 10.4; // compilation error in Java
  - In C++, the compiler usually helps round up or round down the number. However, it will be a compilation error in Java

Expressions

- When writing a program, we use expressions as building blocks, which typically describe actions the program should take
- Two types of expressions
  - Arithmetic expressions
  - Boolean expressions

Arithmetic Expression

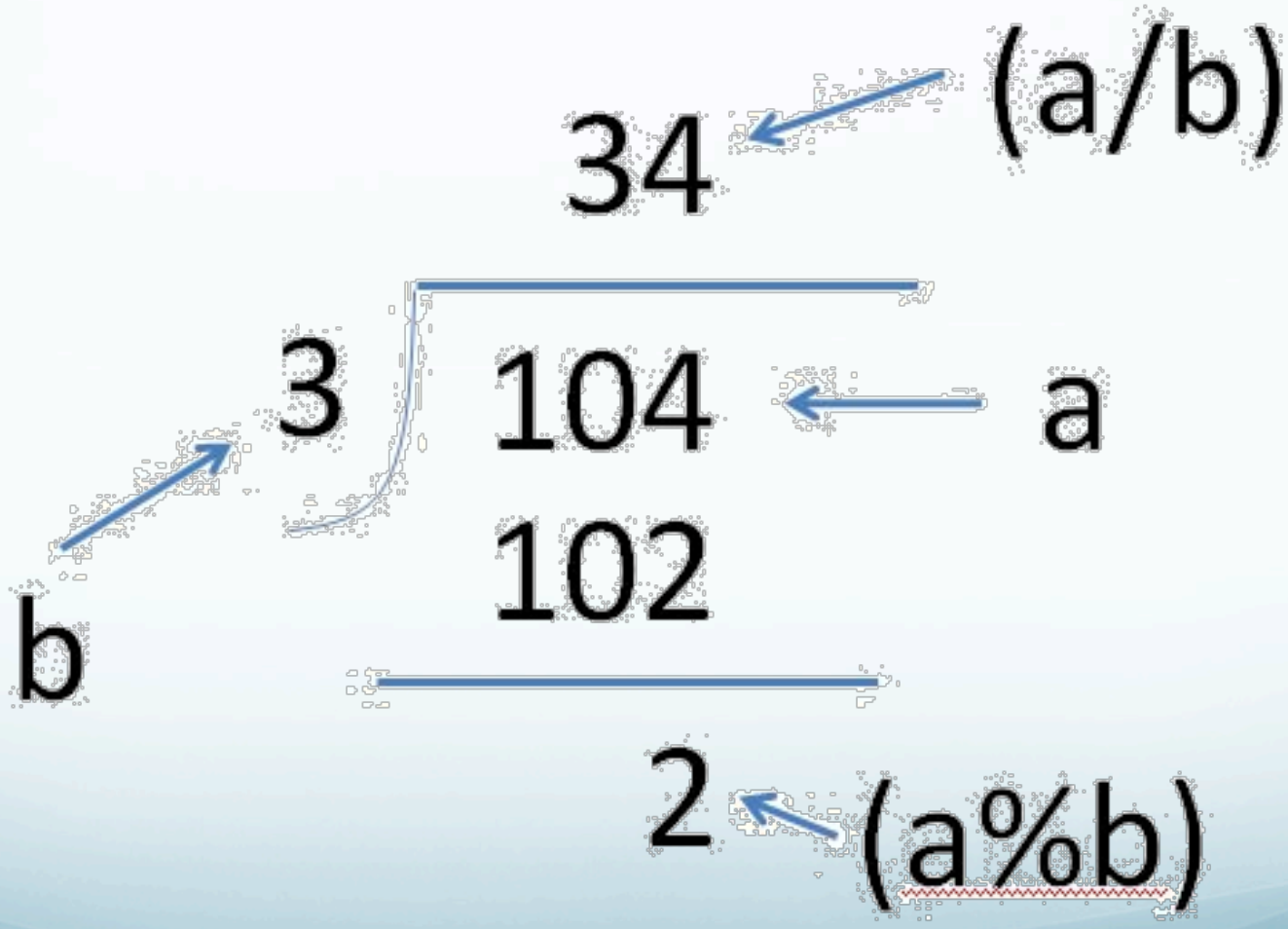
- An arithmetic expression is a sequence of variables, constants, literals connected by arithmetic operators
- Examples
  - An arithmetic expression “2+3” has two literals (2 and 3) connected by a ‘+’ operator
  - An arithmetic expression “celsius*9/5+32” has one variable (celsius), three literals (9, 5 and 32) connected by three operators (*, / and +)

Arithmetic operators

Some commonly used arithmetic operators:

Operator	**Examples**	**Result**
(+) Addition	2 + 3	5
(-) Subtraction	2 – 3	-1
(*) Multiplication	2 * 3	6
(/) Division	2 / 3	0 (Integer division)
(%) Modulus Applicable ONLY if both operands are integers	2 % 3	2 (The remainder of 2 / 3)

Example: Division and Modulus



Integer division and floating-point division

- For data types storing numerical values, we have integers (int) and floating point numbers (double)
- The results are different on division:
  - Integer division
    - Division between two ints
    - Examples: 2/3 results in 0, 3/2 results in 1
  - Floating-point division
    - Division between one or two double's results in a double (e.g. the integer and fraction quotient)
    - Examples: 2/3.0 results in 0.66666, 3.0/2 results in 1.5, 10.0/2.0 results in 5.0

Boolean Expressions

Boolean expressions are similar to arithmetic expressions except:

- Boolean variables only have two possible values: true or false
- Arithmetic operators are now replaced by comparison operators and boolean operators
- Examples:
  - `x == 6` is true if the value stored in x equals to 6, and is false otherwise
  - `7.0 > y` is true if 7.0 is greater than the value stored in y, and is false otherwise

Comparison operators

Operator	Name	Example
<	Less than	2 < 3
<=	Less than or equal to	2 <= 3
>	Greater than	2 > 3
>=	Greater than or equal to	2 >= 3
==	Equal to (Note that, you must have two consecutive ‘=‘)	2 == 3
!=	Not equal to	2 != 3

Boolean operators

- They are used to connect multiple boolean expressions

Operator	Name	Example
!	Logical NOT	! (2 == 3)
&&	Logical AND	4 < 5 && 2 < 3
||	Logical OR	4 < 5 || 2 < 3
^	Exclusive OR	4 < 5 ^ 2 < 3

Truth Table for Logical NOT

- Truth table
  - A mathematical table used to compute the functional values of logical expressions
- Truth table for Logical NOT
  - For example: if **P** represents a false boolean expression, **!P** represents a true boolean expression

P	**!P**
false	true
true	false

Truth Table for Logical AND

- If both **P** and **Q** are true (i.e. both represent true boolean expressions), **P && Q** is true
- Otherwise, **P && Q** is false

P	**Q**	**P && Q**
false	false	false
false	true	false
true	false	false
true	true	true

Truth Table for Logical OR

- If both **P** and **Q** are false (i.e. both represent false boolean expressions), **P || Q** is false
- Otherwise, **P || Q** is true

P	**Q**	**P || Q**
false	false	false
false	true	true
true	false	true
true	true	true

Truth Table for Exclusive OR

- If both **P** and **Q** contain the same truth value,
  P ^ Q is false
- Otherwise, **P ^ Q** is true

P	**Q**	**P ^ Q**
false	false	false
false	true	true
true	false	true
true	true	false

Order of evaluation

How to evaluate:

```
3 + 4 * 4  >  5 * (4 + 3) - 1
```

- Can we simply evaluate from left to right?
- According to our mathematical knowledge, which part should be evaluated first?
  - Operator precedence and associativity govern the evaluation order (see Appendix for details)
- Parentheses can be inserted if you are not sure about (or want to enforce) the evaluation order

Assignment statements and expressions

- Syntax of an assignment statement:
  - Variable = Expression;
    - It means to assign the value evaluated from an expression to the variable
    - The original value stored in that variable (if any) will be replaced
- Example:

```
int result = 0;   // a variable initialized to 0
result = 2 + 3 / 4; // Now, the result becomes 2
```

Short-cuts for assignment operators

There are five short-cut assignment operators

- The expression on the right-hand side of the assignment (=) is evaluated first (the original value of i is used in the calculation, and is then replaced)

Operator	Examples	Equivalent
+=	i += 8	i = i + 8
-=	i -= 8	i = i – 8
*=	i *= 8	i = i * 8
/=	i /= 8	i = i / 8
%=	i %= 8	i = i % 8

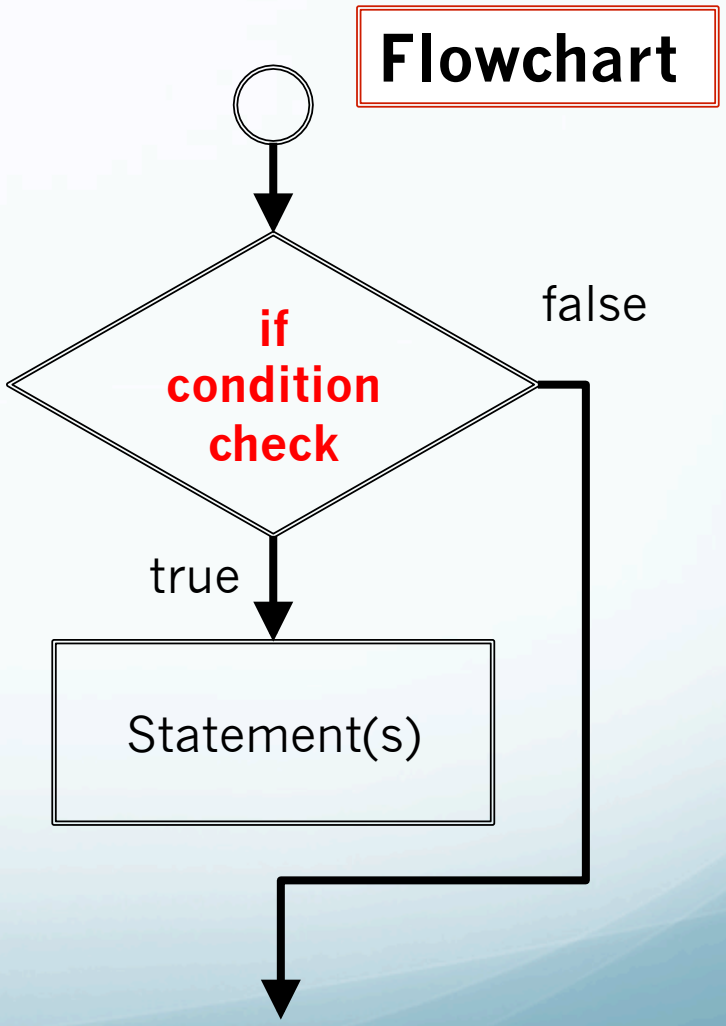
Types of branching statements

- There are three different types of branching statements
  - if statements
  - if-else statements
  - switch statements

If statements

```
if( boolean-expression) {
    // if block
    statement(s);
}
```

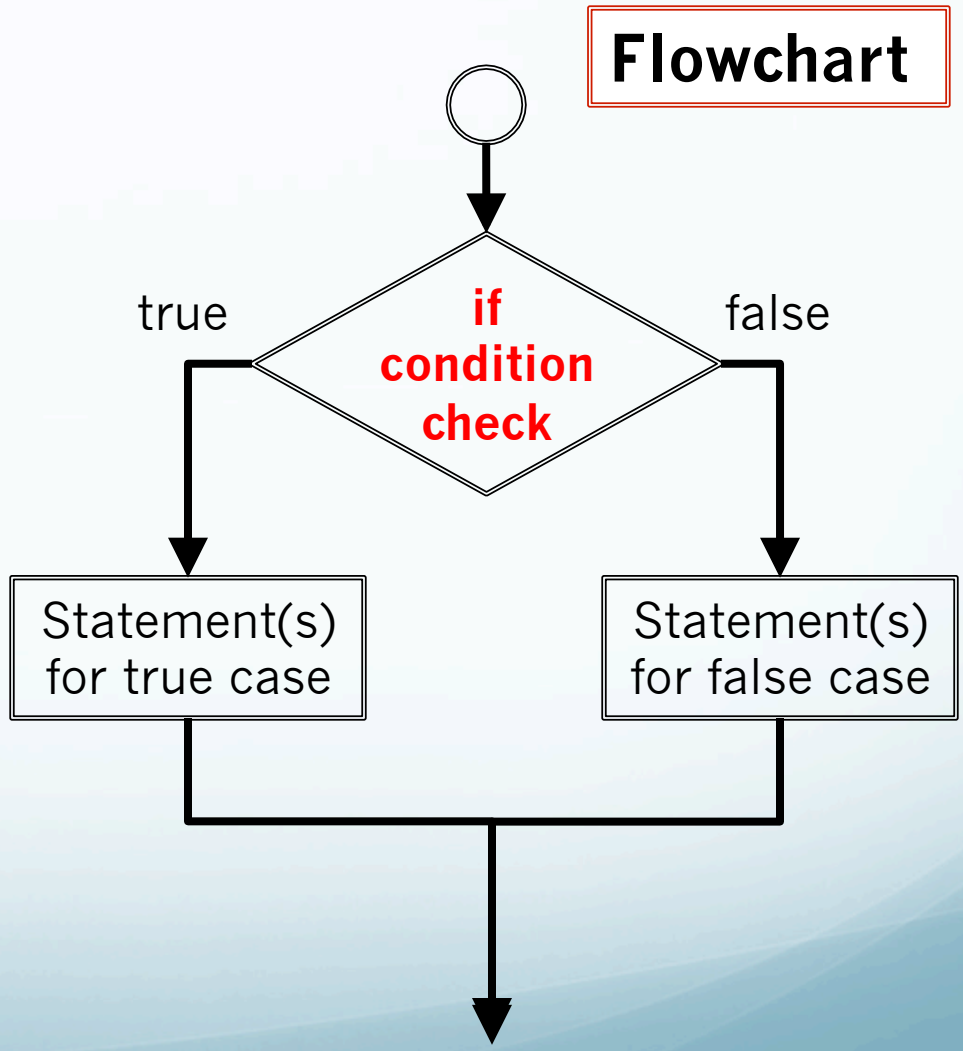
Note: the curly brackets { } can be omitted if there is only one statement



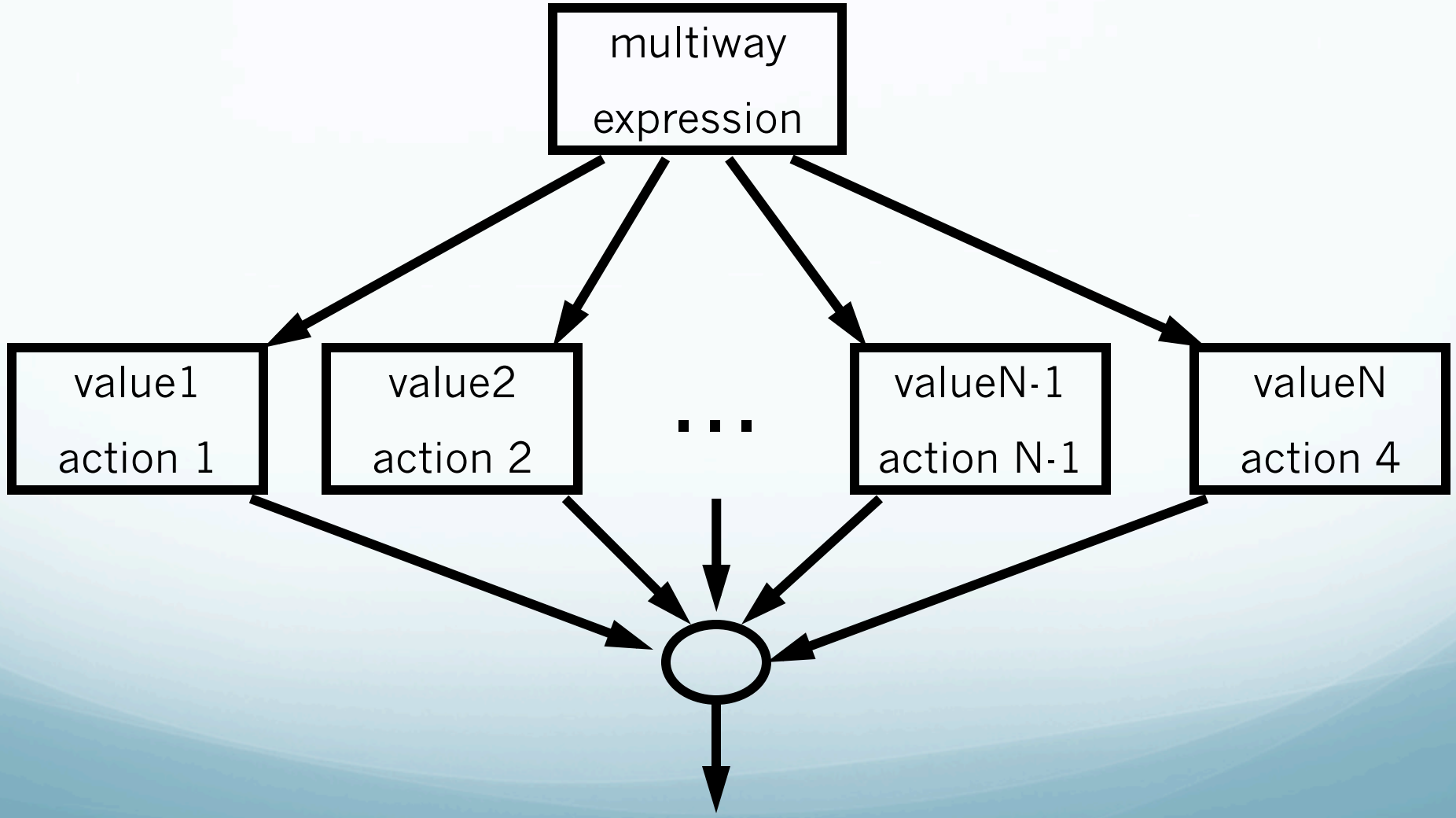
If-else statements

```
if( boolean-expression ) {
  //if-block
  statement(s)-for-true-case;
}
else {
  //else-block
  statements(s)-for-false-case;
}
```

Note: the curly brackets { } can be omitted if there is only one statement in either block



Switch statements



Switch statements

- Switch expression
  - Must be a value of **char**, **byte**, **short**, or **int** type
- The value1, value2...valueN
  - must be of the same type as the switch expression
- The keyword **break** is used to exit the switch statement
  - Without the keyword break, the flow moves to the next case until a break is met
- Default (optional)
  - Only be executed when no other case is matched

```
switch (switch-expression) {
  case value1:  statement(s)1;

 /* If you forget the "break;" here, the
    flow moves to the next case until a
    break is met */
        break;

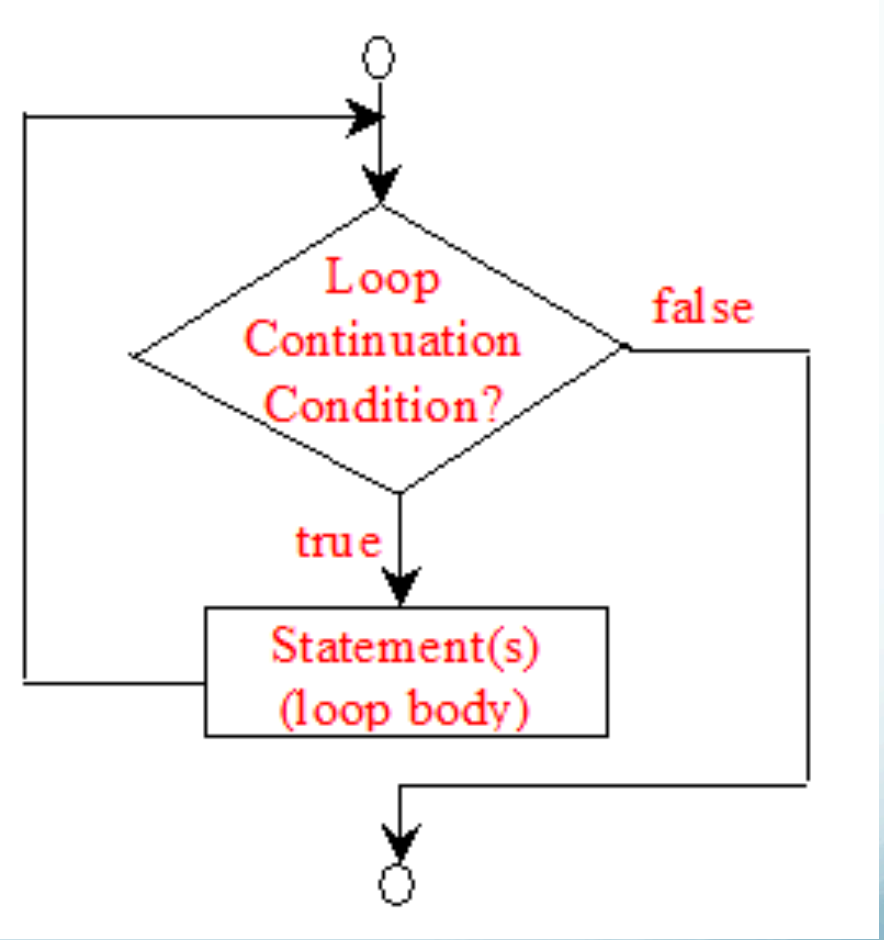
  case value2: statement(s)2;
        break;
 ...
  case valueN: statement(s)N;
        break;
 /* default case is optional */
  default: statement(s)-for-default;
}
```

Loops

- Three types of loops
  - While loop
  - Do-while loop
  - For loop
- Nested Loops
  - Loops can be nested (i.e. an inner loop within the body of an outer loop)

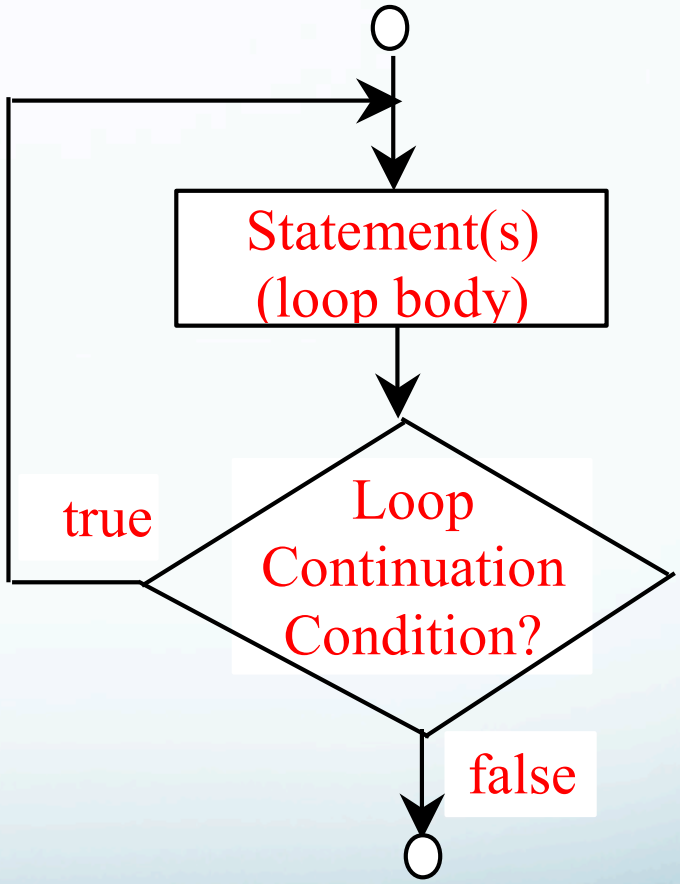
While Loop

```
while (loop-continuation-
    condition)
{
  // loop-body;
  Statement(s);
}
```



Do-while Loop

```
do {
  // Loop body;
  Statement(s);
} while (loop-
  continuation-
  condition);
```

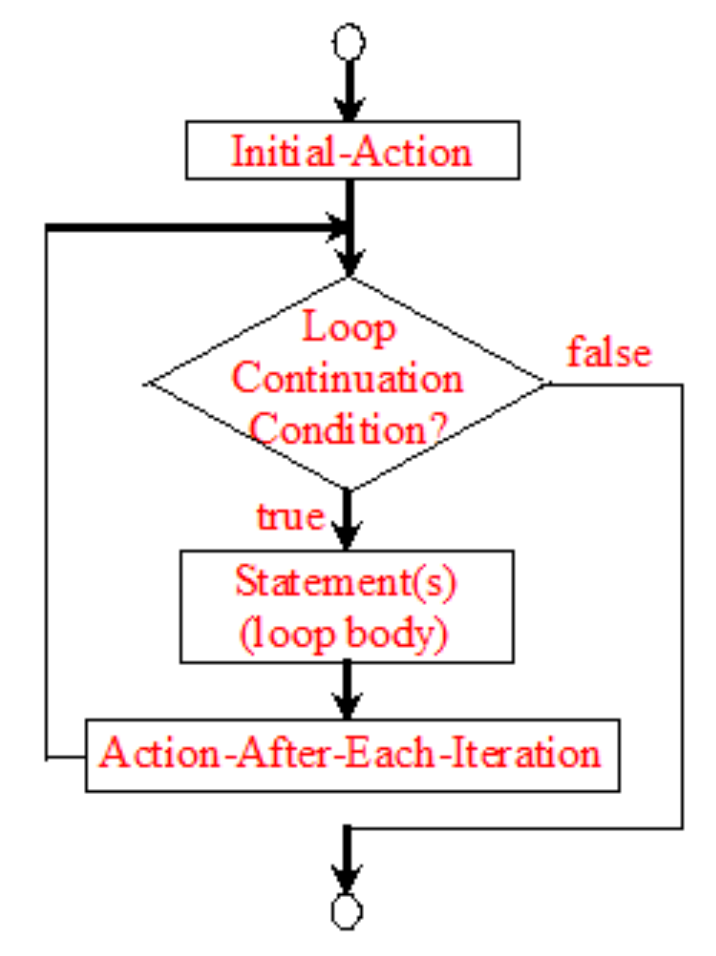


For Loop

```
for (initial-action;

loop-continuation-
    condition;

action-after-each-
    iteration)
{
   // loop body;
   Statement(s);
}
```

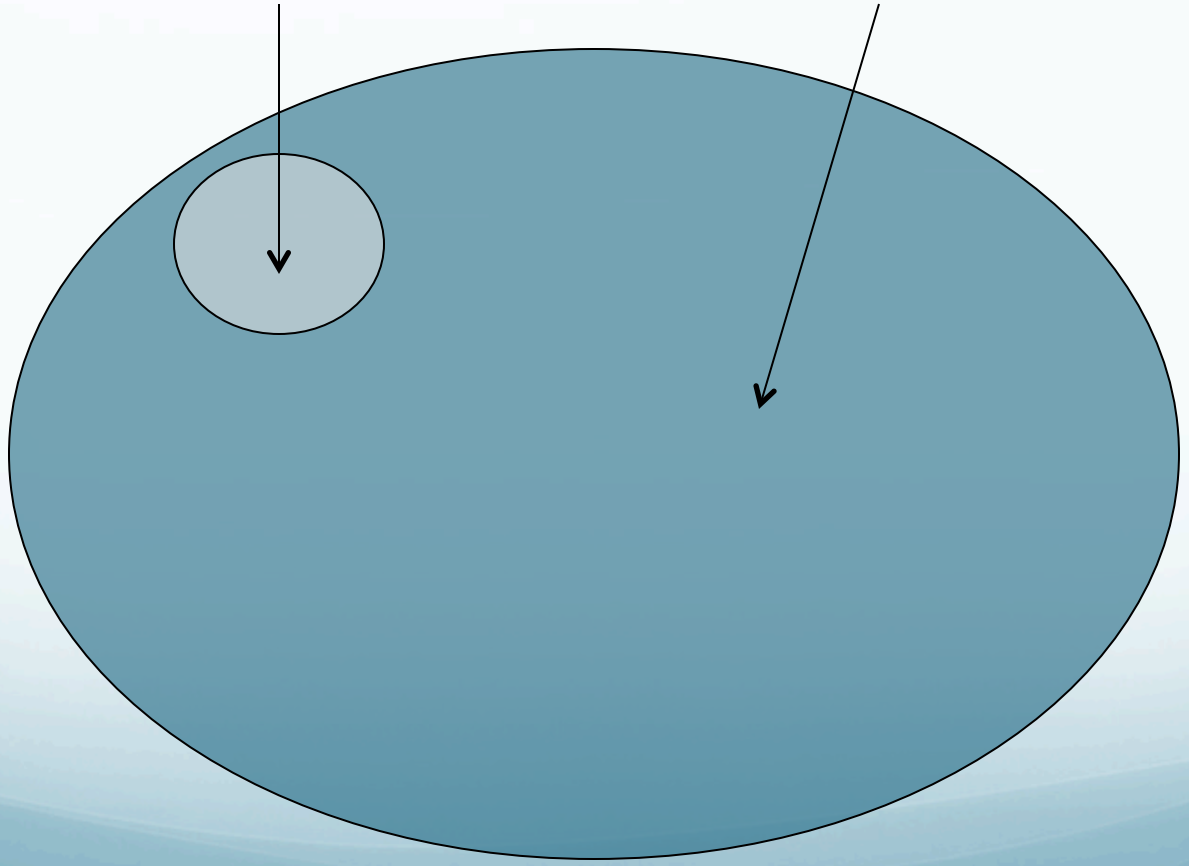


Which Loop Should be Used?

- Programmers are free to choose one of the three loops
- In general
  - While loop
    - The number of repetition is unknown (or unclear), and the loop body may execute 0 to many times
  - Do-while loop
    - The number of repetition is unknown (or unclear), and the loop body must execute at least once
  - For loop
    - The number of repetition is known (e.g. 100 times)

Reference types

- There are two kinds of *types* in the Java programming language: **primitive types** and **reference types**



Reference types

- Unlike C++, Java does not have pointers
- Java uses references to manipulate objects (e.g. String) and arrays
- Example:
  - String s = “Hello World”;
  - In this example, a reference variable s is used to refer a string object storing the content of “Hello World”

String

- String is used to represent a string of characters
  - String message = “Hello World”;
- **String** is actually a predefined class in the Java library just like the **System** class
- The **String** type is not a primitive type. It is known as a ***reference*** type
- String concatenation
  - String message = “Hello" + “World”;

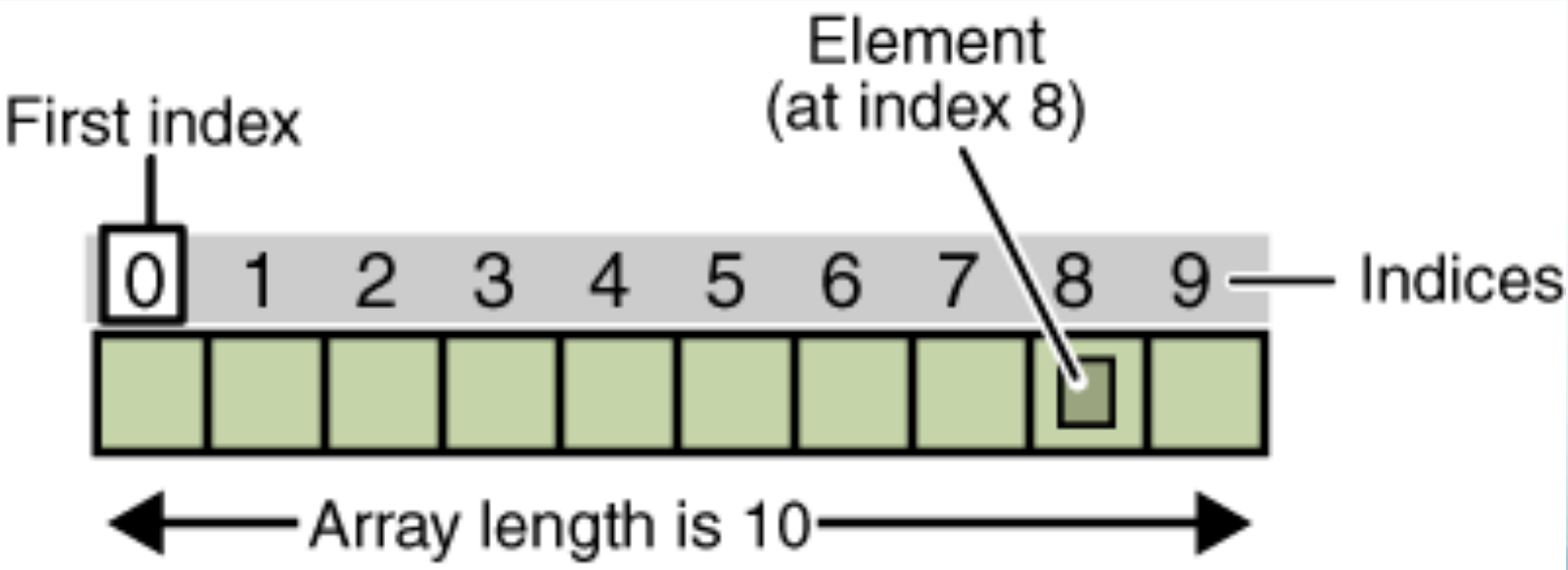
Dynamic allocation

- In Java, all objects are dynamically allocated
  - Examples:
    - C++: string* s = new string("Hello World");
    - Java: String s = new String("Hello World");
- In Java, you are not required to worry about de-allocation
  - The Java Virtual Machine has a garbage collection mechanism to automatically freeing up space in heap
  - Examples:
    - C++: delete s ; // do it my yourself
    - Java: No need to de-allocate memory

Arrays

- Array is a data structure that represents a collection of the same types of data

Example: double[] myList = new double[10];



Creating Arrays

```
arrayRefVar = new datatype[arraySize];
```

Example:

```
double[] myList;
myList = new double[10];
```

`myList[0]` references the first element in the array.

`myList[9]` references the last element in the array.

`myList.length` a built-in field

Creating and initializing an array in one step

- ```
  datatype[] arrayRefVar = new
      datatype[arraySize];
  ```

```
double[] myList = new double[10];
```

```
or double[] myList = new double[]
  {1.0,2.0,3.0};
```

```
String[] args = new String[10];
```
```

Self-learning tutorials

- Oracle (the owner of Java) provides excellent online tutorials to help you learn Java:
  - http://docs.oracle.com/javase/tutorial/