**Java Programming**

*Document version 1.0*

*Programming language version 8.0*



Index

[**Java Introduction** 2](#_Toc12523829)

[**Java Getting Started** 3](#_Toc12523830)

[**Java Syntax** 5](#_Toc12523831)

[**Java Comments** 7](#_Toc12523832)

[**Java Variables** 8](#_Toc12523833)

[**Java Data Types** 11](#_Toc12523834)

[**Java Type Casting** 15](#_Toc12523835)

[**Java Operators** 17](#_Toc12523836)

[**Java Strings** 19](#_Toc12523837)

[**Java Math** 21](#_Toc12523838)

[**Java Booleans** 22](#_Toc12523839)

[**Java If … Else** 23](#_Toc12523840)

[**Java Switch** 25](#_Toc12523841)

[**Java While Loop** 27](#_Toc12523842)

[**Java For Loop** 29](#_Toc12523843)

[**Java Break/Continue** 31](#_Toc12523844)

[**Java Arrays** 33](#_Toc12523845)

[**Java Exceptions – Try … Catch** 36](#_Toc12523846)

[**Java Methods** 38](#_Toc12523847)

# **Java Basic**

## **Java Introduction**

**What is Java?**

Java là ngôn ngữ lập trình phổ biến, được tạo ra từ năm 1995.

Hiện Java thuộc sở hữu của Oracle và hơn 3 tỷ thiết bị chạy Java.

Nó được sử dụng cho:

* Ứng dụng mobile (đặc biệt là ứng dụng Android),
* Ứng dụng desktop,
* Ứng dụng web (front-end & back-end),
* Ứng dụng web server và ứng dụng chạy trên server,
* Games,
* Kết nối cơ sở dữ liệu,
* Và nhiều hơn thế nữa!

**Why Use Java?**

* Java làm việc trên các platform (nền tảng) khác nhau (Windows, Mac, Linux, Raspberry Pi, etc).
* Java là một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới.
* Dễ học và dễ sử dụng.
* Nó là open-source (mã nguồn mở - free) và dễ học.
* Nó bảo mật, nhanh và đầy sức mạnh.
* Nó có một cộng đồng hỗ trợ lớn (hàng chục triệu nhà phát triển).

## **Java Getting Started**

**Java Install**

Latest version: 11 (có tính phí – Oracle)

Link: <https://www.java.com/en/download/win10.jsp>

Recommended Version: **8** update 211

**Check Java Version**

Để kiểm tra phiên bản của Java được cài đặt trên máy tính, sử dụng Command Line (cmd.exe)

|  |
| --- |
| C:\Users\Your Name>java -version |

|  |
| --- |
| java version “1.8.0\_191”  Java(TM) SE Runtime Environment (build 1.8.0\_191-b12)  Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode) |

**Quickstart**

Viết một file Java, MyClass.java, có thể viết bằng một vài trình soạn thảo (Notepad++, Eclipse, NetBean)

MyClass.java

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  System.out.println(“Hello World”);  }  } |

Một cách đơn giản để biên dịch và chạy file MyClass, di chuyển đến thư mục chứa file MyClass.java và gõ câu lệnh bên dưới:

|  |
| --- |
| C:\Users\Your Name>javac MyClass.java |

Gọi chương trình:

|  |
| --- |
| C:\Users\Your Name>java MyClass |

The output should read:

|  |
| --- |
| Hello World1 |

**Development Tool**

Eclipse

NetBean

Notepad++

## **Java Syntax**

**Java Syntax**

Nhắc lại về ví dụ trong chapter trước:

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  System.out.println(“Hello World”);  }  } |

**Giải thích về ví dụ trên**

Mỗi dòng code Java phải ở trong một class. Class được đặt tên là **MyClass**. Một class nên luôn được bắt đầu với một chữ cái viết hoa.

Chú ý: Java là case-sensitive (phân biệt hoa thường, nên MyClass sẽ khác với myclass)

Tên của file Java phải khớp với tên của class. Khi lưu file, lưu nó sử dụng tên class và thêm vào đuôi “.java”.

**Phương thức main**

Phương thức main() là bắt buộc và phải có trong mọi chương trình Java.

|  |  |
| --- | --- |
|  | public static void **main**(String[] args) |

Bất cứ dòng code nào trong phương thức main() đều sẽ được thực thi. Tóm tắt lại, cần phải nhớ những điểm bên dưới:

Mọi chương trình Java đều có tên class

Tên class trùng khớp với tên file

Mọi chương trình phải chứa phương thức main().

**System.out.println()**

Trong phương thức main(), chúng ta có thể sử dụng phương thức println() để in ra dòng chữ trên màn hình:

|  |  |
| --- | --- |
|  | public static void main(String[] args) {  **System.out.println(“Hello World”);**  } |

Chú ý: Mỗi dòng code trong Java sẽ được kết thúc bằng dấu chấm phẩy **;**

## **Java Comments**

Comments có thể được sử dụng để giải thích Java code và làm nó dễ đọc hơn.

Nó có thể được sử dụng để ngăn chặn việc thực thi khi test code.

Comment đơn được bắt đầu bằng dấu //

Các ký tự nằm sau dấu // đến cuối dòng sẽ được bỏ qua khi Java biên dịch.

**Comment dòng đơn**

|  |  |
| --- | --- |
|  | // This is a comment  System.out.println(”Hello World”); |

Comments cũng có thể đặt ở cuối dòng code:

|  |  |
| --- | --- |
|  | System.out.println(”Hello World”); // This is a comment |

**Comment đa dòng**

Bắt đầu bằng /\* và kết thúc bằng \*/.

Tất cả các ký tự nằm trong /\* và \*/ đều sẽ được Java bỏ qua khi biên dịch:

|  |  |
| --- | --- |
|  | /\* The code below will print the words Hello World  to the screen, and it is amazing \*/  System.out.println(”Hello World”); |

## **Java Variables**

**Biến trong Java**

Biến được sử dụng để lưu trữ giá trị dữ liệu.

Trong Java, có nhiều loại giá trị khác nhau, một vài ví dụ:

* String – kiểu chuỗi, giá trị được nằm trong nháy kép “ và “
* int – lưu số nguyên, không chứa phần thập phân, có thể là số âm hoặc dương
* float – lưu số thực, có phần thập phân, ví dụ: 19.99 hoặc -19.99
* char – lưu ký tự đơn, ví dụ như ‘a’ hay ‘B’. Giá trị ký tự nằm trong nháy đơn ‘ và ‘
* boolean – lưu giá trị với 2 trạng thái: **true** hoặc **false**

**Khai báo biến**

Để khai báo biến, cần chỉ rõ kiểu giá trị và gán giá trị cho nó

Cú pháp:

|  |  |
| --- | --- |
|  | *type variable\_name = value;*  *type* : kiểu giá trị, ví dụ int, String, …  *variable\_name* : tên biến, ví dụ: x, studentName, …  *=* : được sử dụng để gán giá trị cho biến  *value* : là giá trị được gán cho biến |

Ví dụ:

|  |  |
| --- | --- |
|  | String name = “John”;  System.out.println(name); |

|  |  |
| --- | --- |
|  | int myNum = 15;  System.out.println(myNum); |

Khai báo biến với một vài kiểu giá trị

|  |  |
| --- | --- |
|  | int myNum = 5;  float myFloatNum = 5.99f;  char myLetter = ‘D’;  boolean myBool = true;  String myText = “Hello”; |

**Hiển thị giá trị biến trên màn hình**

Phương thức println() thường được sử dụng để hiển thị giá trị của biến.

Để kết hợp cả chữ và một biến, sử dụng ký tự +:

|  |  |
| --- | --- |
|  | String name = “John”;  System.out.println(“Hello ” + name); |

Cũng có thể sử dụng ký tự + để cộng nhiều biến lại với nhau:

|  |  |
| --- | --- |
|  | String firstName = “John ”;  String lastName = “Doe”;  String fullName = firstName + lastName;  System.out.println(fullName); |

Đối với biến kiểu số, ký tự + làm việc như là toán tử toán học (trong ví dụ bên dưới, sử dụng kiểu giá trị int).

|  |  |
| --- | --- |
|  | int x = 5;  int y = 6;  System.out.println(x + y); // Print the value of x + y |

**Khai báo nhiều biến**

Để khai báo nhiều hơn một biến cùng một kiểu giá trị, sử dụng dấu phẩy ngăn cách:

|  |  |
| --- | --- |
|  | int x = 5, y = 6, z = 50;  System.out.println(x + y + z); // Print the value of x + y + z |

**Java Identifiers (Bộ định danh – Quy tắc của Java)**

Tất cả các biến trong Java đều phải có tên và không được trùng nhau.

Các tên này được gọi là bộ định danh (**Identifiers**).

Bộ định danh có thể là ngắn hoặc dài là tùy ý.

Các quy định chung cho việc đặt tên biến (định danh):

* Tên có thể chứa ký tự, số, dấu gạch dưới, và ký tự **$**
* Tên nên bắt đầu với một ký tự
* Tên có thể bắt đầu bằng ký tự **$** và dấu gạch dưới **\_**
* Tên là phân biệt hoa thường (myVar khác với myvar)
* Tên nên bắt đầu với một ký tự viết thường và nó không thể chứa khoảng trắng
* Các từ khóa không được phép sử dụng (ví dụ: int, String, …)

## **Java Data Types**

**Java Data Types**

Một biến trong Java phải được xác định bởi một kiểu giá trị (data type):

|  |  |
| --- | --- |
|  | int myNum = 5; // Integer (whole number)  float myFloatNum = 5.99f; // Floating point number  char myLetter = ‘D’; // Character  boolean myBool = true; // Boolean  String myText = “Hello”; // String |

Kiểu giá trị được chia làm 2 nhóm:

* ***Kiểu dữ liệu nguyên thủy*** (primitive data types) – bao gồm: byte, short, int, long, float, double, boolean và char
* ***Kiểu dữ liệu không nguyên thủy*** (non-primitive data types) – như là String, Arrays và Classes.

**Kiểu dữ liệu nguyên thủy (Primitive Data Types)**

Một kiểu dữ liệu nguyên thủy chỉ định rõ kích thước và loại giá trị biến và nó không có phương thức bổ sung.

Có 8 loại kiểu dữ liệu nguyên thủy trong Java:

|  |  |  |
| --- | --- | --- |
| Kiểu dữ liệu | Kích thước | Mô tả |
| byte | 1 byte | -128 ~ 127 |
| short | 2 bytes | -32,768 ~ 32,767 |
| int | 4 bytes | -2,147,483,648 ~ 2,147,483,647 |
| long | 8 bytes | -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 |
| float | 4 bytes | Lưu trữ số phân số, đủ để lưu trữ 6 ~ 7 ký tự thập phân |
| double | 8 bytes | Lưu trữ số phân số, đủ để lưu trữ 15 ký tự thập phân |
| boolean | 1 bit | Lưu trữ 2 giá trị true, false |
| char | 2 bytes | Lưu trữ một ký tự đơn/chữ đơn hoặc giá trị ASCII |

**Numbers**

Loại số nguyên thủy được chia làm 2 nhóm:

**Kiểu số nguyên (*Integer types*)** chứa số nguyên, số dương hoặc số âm. Kiểu giá trị hợp lệ là byte, short, int và long.

**Kiểu giá trị số thực (*Floating point types*)** biểu thị số thập phân, chứa một hoặc nhiều ký tự thập phân. Có 2 loại: float và double.

**Integer Types**

Byte

Kiểu byte có thể lưu trữ dải số từ -128 đến 127.

|  |  |
| --- | --- |
|  | byte myNum = 100;  System.out.println(myNum); |

Short

Kiểu short có thể chứa dải số từ -32768 đến 32767

|  |  |
| --- | --- |
|  | short myNum = 5000;  System.out.println(myNum); |

Int

Kiểu int có thể chứa dải số từ -2147483648 đến 2147483647.

|  |  |
| --- | --- |
|  | int myNum = 100000;  System.out.println(myNum); |

Long

Kiểu long có thể chứa dải số từ -9223372036854775808 đến 9223372036854775807.

|  |  |
| --- | --- |
|  | long myNum = 15000000000L;  System.out.println(myNum); |

**Floating Point Types**

Có thể sử dụng kiểu số thực những lúc cần chứa số thập phân.

Float

Kiểu float có thể chứa dải giá trị từ 3.4e-038 đến 3.4e+038. Kết thúc giá trị luôn có chữ f.

|  |  |
| --- | --- |
|  | float myNum = 19.99f;  System.out.println(myNum); |

Double

Kiểu double có thể chứa dải số từ 1.7e-308 đến 1.7e+308. Sau giá trị có chữ d.

|  |  |
| --- | --- |
|  | double myNum = 19.99d;  System.out.println(myNum); |

**Scientific Numbers**

Một số thực có thể là một số khoa học với một chữ “e” để xác định lũy thừa của 10.

|  |  |
| --- | --- |
|  | float f1 = 35e3f;  double d1 = 12E4d;  System.out.println(f1);  System.out.println(d1); |

**Booleans**

Một giá trị kiểu boolean chỉ có thể chứa 1 trong 2 giá trị là true hoặc false.

|  |  |
| --- | --- |
|  | boolean isJavaFun = true;  boolean isFishTasty = false;  System.out.println(isJavaFun); // Outputs true  System.out.println(isFishTasty); // Outputs false |

**Characters**

Giá trị kiểu char được sử dụng để lưu trữ một ký tự đơn. Giá trị được đặt trong dấu nháy đơn.

|  |  |
| --- | --- |
|  | char myGrade = ‘B’;  System.out.println(myGrade); |

Ngoài ra, có thể sử dụng giá trị ASCII để hiển thị các ký tự nhất định:

|  |  |
| --- | --- |
|  | char a = 65, b = 66, c = 67;  System.out.println(a);  System.out.println(b);  System.out.println(c); |

**Strings**

Kiểu string được sử dụng để lưu một chuỗi ký tự. Giá trị nằm trong dấy nháy kép.

|  |  |
| --- | --- |
|  | String greeting = “Hello World”;  System.out.println(greeting); |

**Kiểu dữ liệu không nguyên thủy (Non-Primitive Data Types)**

Kiểu dữ liệu này còn được gọi là kiểu dữ liệu tham chiếu bởi vì nó tham khảo đến các đối tượng.

Sự khác biệt chính giữa kiểu nguyên thủy và không nguyên thủy là:

* Kiểu nguyên thủy đều đã được định nghĩa trước trong Java. Kiểu không nguyên thủy được tạo ra bởi người lập trình (ngoại trừ String).
* Kiểu không nguyên thủy có thể được sử dụng để gọi các phương thức nhất định, trong khi kiểu nguyên thủy thì không thể.
* Kiểu nguyên thủy luôn có giá trị, trong khi kiểu không nguyên thủy thì có thể null.
* Kiểu nguyên thủy bắt đầu với một chữ cái thường, trong khi kiểu không nguyên thủy bắt đầu bằng một chữ cái in hoa.
* Kích thước của một kiểu nguyên thủy phụ thuộc vào kiểu dữ liệu, trong khi các kiểu không nguyên thủy đều có cùng kích thước.

## **Java Type Casting**

**Java Type Casting**

Ép kiểu được thực hiện khi gán một giá trị của một kiểu dữ liệu nguyên thủy tới một kiểu khác.

Trong Java, có 2 loại ép kiểu:

* ***Widening Casting*** (tự động) – chuyển đổi một kiểu có kích thước nhỏ hơn thành một kiểu có kích thước lớn hơn.

byte -> short -> char -> int -> long -> float -> double

* ***Narrowing Casting*** (bằng tay) – chuyển đổi kiểu có kích thước lớn hơn về kiểu có kích thước nhỏ hơn.

double -> float -> long -> int -> char -> short -> byte

**Widening Casting**

Ép kiểu *widening* được thực hiện một cách tự động khi chuyển một kiểu có kích thước nhỏ hơn đến một kiểu có kích thước lớn hơn.

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  int myInt = 9;  **double myDouble = myInt;** // Automatic castinig: int to double    System.out.println(myInt); // Outputs 9  System.out.println(myDouble); // Outputs 9.0  }  } |

**Narrowing Casting**

Ép kiểu *narrowing* phải được thực hiện bằng tay bằng việc đưa vào dấu ngoặc đơn cho kiểu được thay thế đứng đằng trước giá trị:

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  dounle myDouble = 9.78;  int myInt = (int) myDouble; // Manual casting: double to int    System.out.println(myDouble); // Outputs 9.78  System.out.println(myInt); // Outputs 9  }  } |

## **Java Operators**

Toán tử được sử dụng để biểu diễn các hoạt động diễn ra trên các biến và giá trị.

Giá trị được gọi là operand (toán hạng), trong khi hoạt động được định nghĩa là operator (điều hành).

|  |  |  |
| --- | --- | --- |
| Operand | Operator | Operand |
| 100 | + | 50 |

Ví dụ:

|  |  |
| --- | --- |
|  | int sum1 = 100 + 50; // 150 (100 + 50)  int sum2 = sum1 + 250; // 400 (150 + 250)  int sum3 = sum2 + sum2; // 800 (400 + 400) |

Java phân chia các toán tử thành các nhóm như bên dưới:

* Arithmetic operators (số học)
* Assignment operators (gán)
* Comparison operators (so sánh)
* Logical operators (logic)
* Bitwise operators (bitwise)

**Java Arithmetic Operators**

Toán tử số học được sử dụng để biểu diễn các phép toán phổ biến.

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| + | Addition | Cộng 2 giá trị | x + y |
| - | Subtraction | Trừ một giá trị cho một giá trị | x – y |
| \* | Multiplication | Nhân 2 giá trị | x \* y |
| / | Division | Chia một giá trị cho một giá trị | x / y |
| % | Modulus | Trả về phần dư | x % y |
| ++ | Increment | Tăng giá trị của biến lên 1 | ++x |
| -- | Decrement | Giảm giá trị của biến đi 1 | --x |

**Java Assignment Operators**

Toán tử gán được sử dụng để gián giá trị cho biến.

Ví dụ:

|  |  |
| --- | --- |
|  | int x = 10;  x += 5; |

Danh sách các toán tử gán:

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same as |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x – 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

**Java Comparision Operators**

Các toán tử so sánh được sử dụng để so sánh 2 giá trị:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## **Java Strings**

**Java Strings**

Kiểu chuỗi được sử dụng để lưu trữ văn bản.

Một biến kiểu chuỗi chứa một tập hợp các ký tự và được đặt trong dấu nháy kép.

|  |  |
| --- | --- |
|  | String greeting = “Hello”; |

**String Length (Chiều dài chuỗi)**

Để lấy chiều dài chuỗi, sử dụng phương thức length().

|  |  |
| --- | --- |
|  | String txt = “ABCDEFGHIJKLMNOPQRSTUVWXYZ”;  System.out.println(“The length of the text string is: ” + txt.length()); |

**More String Methods (Các phương thức khác)**

Có nhiều phương thức dành cho String, ví dụ toUpperCase() và toLowerCase():

|  |  |
| --- | --- |
|  | String txt = “Hello World”;  System.out.println(txt.toUpperCase());  System.out.println(txt.toLowerCase()); |

**Finding a String in a String (Tìm chuỗi trong chuỗi)**

Phương thức indexOf() được sử dụng để trả về index (chỉ số) của giá trị đầu tiên xuất hiện trong một chuỗi (bao gồm cả khoảng trắng):

|  |  |
| --- | --- |
|  | String txt = “Please locate where ‘locate’ occurs!”;  System.out.println(txt.indexOf(“locate”)); |

Chú ý: Java đến vị trí tính từ 0. 0 là vị trí đầu tiên trong một chuỗi, tiếp sau là 1, 2, …

**String Concatenation (Cộng chuỗi)**

Toán tử + được sử dụng giữa 2 chuỗi để cộng chúng lại với nhau và tạo ra một chuỗi mới.

|  |  |
| --- | --- |
|  | String firstName = “John”;  String lastName = “Doe”;  System.out.println(firstName + “” + lastName); |

Hoặc sử dụng phương thức concat() để cộng chuỗi:

|  |  |
| --- | --- |
|  | String firstName = “John”;  String lastName = “Doe”;  System.out.println(firstName.concat(lastName)); |

**Special Characters (Ký tự đặc biệt)**

Bởi vì giá trị chuỗi được viết trong dấu nháy kép, nên nó sẽ không hiểu được trong trường hợp này:

|  |  |
| --- | --- |
|  | String txt = “We are the so-called “Vikings” from the north.”; |

Để giải quyết vấn đề này, chúng ta sử dụng ký tự backslash (\)

|  |  |  |
| --- | --- | --- |
| Escape character | Result | Description |
| \’ | ‘ | Nháy đơn |
| \” | “ | Nháy kép |
| \\ | \ | Backslash |

|  |  |
| --- | --- |
|  | String txt = “We are the so-called \“Vikings\” from the north.”; |

Sáu chuỗi escape khác hợp lệ trong Java:

|  |  |  |
| --- | --- | --- |
| Escape character | Result |  |
| \n | Xuống dòng |  |
| \r | Carriage return – vận chuyển trở lại |  |
| \t | Tab |  |
| \b | Backspace |  |
| \f | Form Feed |  |

## **Java Math**

Java Math class có nhiều phương thức cho phép biểu diễn các phép toán trên chữ số.

**Math.max(x, y)**

Phương thức này tìm ra số lớn nhất trong 2 số x và y

|  |  |
| --- | --- |
|  | Math.max(5, 10); |

**Math.min(x, y)**

Phương thức này tìm ra số nhỏ nhất trong 2 số x và y

|  |  |
| --- | --- |
|  | Math.min(5, 10); |

**Math.sqrt(x)**

Phương thức này trả về kết quả của căn bậc 2 của x

|  |  |
| --- | --- |
|  | Math.sqrt(64); |

**Math.abs(x)**

Phương thức này trả về giá trị tuyệt đối của x

|  |  |
| --- | --- |
|  | Math.abs(-4.7); |

**Math.random()**

Phương thức này trả về số ngẫu nhiên nằm giữa 0 và 1

|  |  |
| --- | --- |
|  | Math.random(); |

## **Java Booleans**

**Java Booleans**

Trong lập trình thì việc cần một loại dữ liệu chỉ có 2 giá trị là cần thiết và được sử dụng thường xuyên:

* YES/ NO
* ON/ OFF
* TRUE/ FALSE

Java có kiểu boolean, có 2 giá trị true và false.

**Boolean Values**

Ví dụ:

|  |  |
| --- | --- |
|  | boolean isJavaFun = true;  boolean isFishTasty = false;  System.out.println(isJavaFun);  System.out.println(isFishTasty); |

**Boolean Expression (Biểu thức)**

Một biểu thức boolean là biểu thức Java trả về giá trị Boolean: true hoặc false.

|  |  |
| --- | --- |
|  | int x = 10;  int y = 9;  System.out.println(x > y); // returns true |

|  |  |
| --- | --- |
|  | int x = 10;  System.out.println(x == 10); // returns true |

## **Java If … Else**

**Java Conditions and If Statements (Điều kiện và cú pháp If)**

Java hỗ trợ điều kiện logic từ toán học:

* Nhỏ hơn: a < b
* Nhỏ hơn hoặc bằng: a <= b
* Lớn hơn: a > b
* Lớn hơn hoặc bằng: a >= b
* Bằng: a == b
* Không bằng: a != b

Java có các cú pháp điều kiện như bên dưới:

* if: chỉ ra block được thực thi nếu điều kiện là true
* else: chỉ ra block được thực thi nếu điều kiện là false
* else if: chỉ ra điều kiện mới để test, nếu điều kiện đầu là false
* switch: chỉ ra nhiều block được thực thi dựa theo điều kiện

**Syntax (Cú pháp)**

|  |  |
| --- | --- |
|  | if (condition) {  // block of code to be executed if the condition is true  } |

**Else Statement (Cú pháp Else)**

Sử dụng else để chỉ ra khối code được thực thi nếu điều kiện ở trên nhánh if là false.

|  |  |
| --- | --- |
|  | if (condition) {  // block of code to be executed if the condition is **true**  } else {  // block of code to be executed if the condition is **false**  } |

Ví dụ:

|  |  |
| --- | --- |
|  | int time = 20;  if (time < 18) {  System.out.println(“Good day.”);  } else {  System.out.println(“Good evening.”);  } |

**Else if Statement (Cú pháp else if)**

Sử dụng else if để chỉ ra điều kiện mới nếu điều kiện đầu là false.

|  |  |
| --- | --- |
|  | if (condition1) {  // block of code to be executed if the condition1 is **true**  } else if (condition2) {  // block of code to be executed if the condition1 is **false** and condition2 is **true**  } else {  // block of code to be executed if the condition1 is **false** and condition2 is **false**  } |

**Short Hand If…Else (Cú pháp If else ngắn)**

Nếu bạn có một cú pháp để thực thi mà chỉ có if và else thì có thể viết trên cùng 1 dòng như sau:

|  |  |
| --- | --- |
|  | variable = (condition) ? expressionTrue : expressionFalse |

Ví dụ:

|  |  |
| --- | --- |
|  | int time = 20;  String result = (time < 18) ? “Good day.” : “Good evening.”; |

## **Java Switch**

**Java Switch Statements (Cú pháp switch)**

Sử dụng switch để lựa chọn một trong số nhiều khối lệnh được thực hiện.

|  |  |
| --- | --- |
|  | switch (expression) {  case x:  // code block  break;  case y:  // code block  break;  default:  // code block  } |

*Cách hoạt động:*

* Biểu thức switch được đánh giá 1 lần.
* Giá trị của biểu thức switch được so sánh với giá trị của mỗi case.
* Nếu có trường hợp khớp, thì khối lệnh tương ứng sẽ được thực thi.
* Hai từ khóa break và default là optional (không bắt buộc).

Ví dụ:

|  |  |
| --- | --- |
|  | int day = 4;  switch (day) {  case 1:  System.out.println(“Monday”);  break;  case 2:  System.out.println(“Tuesday”);  break;  case 3:  System.out.println(“Wednesday”);  break;  case 4:  System.out.println(“Thursday”);  break;  case 5:  System.out.println(“Friday”);  break;  case 6:  System.out.println(“Saturday”);  break;  case 7:  System.out.println(“Sunday”);  break;  }  // Outputs “Thursday” (day 4) |

**Break Keyword (từ khóa break)**

Khi Java gặp từ khóa break, nó sẽ thoát khỏi switch.

## **Java While Loop**

**Loops (Vòng lặp)**

Vòng lặp có thể được sử dụng để thực thi 1 khối dòng lệnh theo một điều kiện cụ thể nào đó.

**Java While Loop**

Vòng lặp while lặp qua một khối mã lệnh miễn là điều kiện được chỉ định là true.

Syntax (cú pháp)

|  |  |
| --- | --- |
|  | while (*condition*) {  // code block to be executed  } |

Ví dụ

|  |  |
| --- | --- |
|  | int i = 0;  while (i < 5) {  System.out.println(i);  i++;  } |

**Vòng lặp Do/While**

Là một dạng khác của vòng lặp while. Vòng lặp này sẽ thực thi khối lệnh 1 lần, trước khi nó kiểm tra điều kiện là true, sau đó nó lặp lại việc thực thi khối lệnh khi điều kiện vẫn là true.

Syntax (cú pháp)

|  |  |
| --- | --- |
|  | do {  // code block to be executed  }  while (*condition*); |

Ví dụ

|  |  |
| --- | --- |
|  | int i = 0;  do {  System.out.println(i);  i++;  }  while (i < 5); |

## **Java For Loop**

**Java For Loop (Vòng lặp For)**

Khi bạn biết chính xác số lần lặp qua một khối lệnh, hãy sử dụng vòng lặp for thay vì while.

Syntax (cú pháp)

|  |  |
| --- | --- |
|  | for (*statement 1; statement 2; statement 3*) {  // code block to be executed  } |

**Statement 1**: được thực hiện (1 lần) trước khi khối mã lệnh được thực hiện

**Statement 2**: định nghĩa điều kiện để thực thi khối mã lệnh

**Statement 3**: được thực hiện (mỗi lần) sau khi khối mã lệnh được thực thi

Ví dụ:

|  |  |
| --- | --- |
|  | for (int i = 0; i < 5; i++) {  System.out.println(i);  } |

Một ví dụ khác:

|  |  |
| --- | --- |
|  | for (int i = 0; i <= 10; i = i + 2) {  System.out.println(i);  } |

**For-Each Loop (Vòng lặp For-Each)**

Có một vòng lặp “**for-each**”, được sử dụng riêng để lặp qua các phần tử của mảng (**array**):

Syntax (cú pháp)

|  |  |
| --- | --- |
|  | for (*type variable : arrayname*) {  // code block to be executed  } |

Ví dụ

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Madza”};  for (String i : cars) {  System.out.println(i);  } |

## **Java Break/Continue**

**Java Break**

Bạn đã thấy cú pháp break được sử dụng trong switch.

Cú pháp break cũng có thể được sử dụng trong vòng lặp.

Ví dụ:

|  |  |
| --- | --- |
|  | for (int i = 0; i < 10; i++) {  if (i == 4) {  break;  }  System.out.println(i);  } |

**Java Continue**

Cú pháp continue phá vỡ một lần lặp (trong vòng lặp), nếu một điều kiện chỉ định được xảy ra và tiếp tục với lần lặp tiếp theo trong vòng lặp.

Ví dụ:

|  |  |
| --- | --- |
|  | for (int i = 0; i < 10; i++) {  if (i == 4) {  continue;  }  System.out.println(i);  } |

**Break and Continue in While Loop**

Bạn có thể sử dụng break và continue trong vòng lặp while.

Ví dụ break:

|  |  |
| --- | --- |
|  | int i = 0;  while (i < 10) {  System.out.println(i);  i++;  if (i == 4) {  break;  }  } |

Ví dụ continue:

|  |  |
| --- | --- |
|  | int i = 0;  while (i < 10) {  if (i == 4) {  i++;  continue;  }  System.out.println(i);  i++;  } |

## **Java Arrays**

**Java Arrays**

Mảng được sử dụng để lưu trữ nhiều giá trị trong chỉ 1 biến, thay vì phải khai báo biến cho mỗi một giá trị.

Để khai báo mảng, định nghĩa kiểu giá trị với dấu ngoặc vuông:

|  |  |
| --- | --- |
|  | String[] cars; |

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”}; |

|  |  |
| --- | --- |
|  | int[] myNum = {10, 20, 30, 40}; |

**Access the Elements of an Array (Truy cập phần tử mảng)**

Bạn có thể truy cập vào phần tử của mảng thông qua chỉ số (index), mảng trong Java có chỉ số bắt đầu từ 0 ứng với vị trí đầu tiên trong mảng.

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};  System.out.println(cars[0]); |

**Change an Array Element (Thay đổi một phần tử của mảng)**

Để thay đổi giá trị của một phần tử trong mảng, bạn tham khảo đến số index.

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};  cars[0] = “Opel”;  System.out.println(cars[0]);  // Now ouputs Opel instead of Volvo |

**Array Length (chiều dài mảng)**

Sử dụng thuộc tính length để trả về chiều dài của mảng.

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};  System.out.println(cars.length);  // Ouputs 4 |

**Loop Through an Array**

Sử dụng vòng lặp for và kết hợp với thuộc tính length của mảng.

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};  for (int i = 0; i < cars.length; i++) {  System.out.println(cars[i]);  } |

**Loop Through an Array with For-Each**

Hoặc có thể sử dụng for-each để duyệt qua các giá trị của mảng.

|  |  |
| --- | --- |
|  | String[] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};  for (String i : cars) {  System.out.println(i);  } |

**Multidimensional Arrays (Mảng đa chiều)**

Một mảng đa chiều là một mảng chứa một hoặc nhiều mảng khác.

Để tạo mảng 2 chiều, thêm từng mảng trong dấu ngoặc nhọn.

Ví dụ:

|  |  |
| --- | --- |
|  | int[][] myNumbers = {{1, 2, 3, 4}, {5, 6, 7}}; |

Truy xuất giá trị một phần tử trong mảng 2 chiều thông qua chỉ số hàng và cột.

|  |  |
| --- | --- |
|  | int[][] myNumbers = {{1, 2, 3, 4}, {5, 6, 7}};  int x = myNumbers[1][2];  System.out.println(x); // Outputs 7 |

Sử dụng vòng lặp for để duyệt các giá trị trong mảng 2 chiều:

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  int[][] myNumbers = {{1, 2, 3, 4}, {5, 6, 7}};  for (int i = 0; i < myNumbers.length; ++i) {  for (int j = 0; j < myNumbers[i].length; ++j) {  System.out.println(myNumbers[i][j]);  }  }  }  } |

## **Java Exceptions – Try … Catch**

**Java Exceptions**

Khi thực thi mã lệnh, các lỗi khác nhau có thể xảy ra là: lỗi coding từ phía lập trình viên, lỗi do nhập sai dữ liệu hoặc những lỗi không lường trước được.

Khi có lỗi xảy ra, Java sẽ dừng chương trình và tạo thông báo lỗi. Thuật ngữ kỹ thuật cho trường hợp này là: Java will throw an exception (throw an error).

**Java try and catch**

Khối try cho phép định nghĩa một khối mã lệnh được kiểm tra lỗi trong khi nó được thực thi.

Khối catch cho phép bạn định nghĩa một khối mã lệnh được thực thi nếu có lỗi xảy ra trong khối try.

Cú pháp:

|  |  |
| --- | --- |
|  | try {  // Block of code to try  }  catch(Exception *e*) {  // Block of code to handle errors  } |

Ví dụ:

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  try {  int[] myNumbers = {1, 2, 3};  System.out.println(myNumbers[10]);  } catch (Exception e) {  System.out.println(“Some thing went wrong”);  }  }  } |

**Finally**

Câu lệnh finally giúp bạn thực thi mã lệnh, sau try…catch, bất kể kết quả.

|  |  |
| --- | --- |
|  | public class MyClass {  public static void main(String[] args) {  try {  int[] myNumbers = {1, 2, 3};  System.out.println(myNumbers[10]);  } catch (Exception e) {  System.out.println(“Some thing went wrong”);  } finally {  System.out.println(“The ‘try catch’ is finished”);  }  }  } |

**The throw keyword**

Câu lệnh throw cho phép bạn tạo một lỗi tùy chỉnh.

Câu lệnh throw được sử dụng cùng với một exception type (ArithmeticException, ClassNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc.)

|  |  |
| --- | --- |
|  | public class MyClass {  static void checkAge(int age) {  if (age < 18) {  throw new ArithmeticException(“Access denied – You must be at least 18 years old.”);  System.out.println(myNumbers[10]);  } else {  System.out.println(“Access granted – You are old enough!”);  }  }  public static void main(String[] args) {  checkAge(15); // Set age to 15 (which is below 18 …)  }  } |

## **Java Methods**

Một method là một khối mã lệnh và chỉ được thực thi khi được gọi đến.

Bạn có thể truyền dữ liệu, được gọi là tham số, vào trong phương thức.

Các phương thức được sử dụng để thực hiện các hành động nhất định, được gọi là các hàm.

Tại sao phải sử dụng method? Để tái sử dụng code: định nghĩa code một lần, và sử dụng nó nhiều lần.

**Create a Method**

Tạo method trong class.

|  |  |
| --- | --- |
|  | public class MyClass {  static void **myMethod()** {  // code to be executed  }  } |

**Call a Method**

Để gọi một phương thức trong Java, viết tên phương thức với dấu **()** và ký tự **;**

|  |  |
| --- | --- |
|  | public class MyClass {  static void myMethod() {  System.out.println(“I just got executed”);  }  public static void main(String[] args) {  **myMethod();**  }  }  // Outputs “I just got executed!” |

**Method Parameters (Tham số)**

Tham số được xác định sau tên phương thức. Một phương thức có thể có nhiều tham số.

|  |  |
| --- | --- |
|  | public class MyClass {  static void myMethod(**String fname**) {  System.out.println(fname + “ Refsnes”);  }  public static void main(String[] args) {  myMethod(“Liam”);  myMethod(“Jenny”);  myMethod(“Anja”);  }  }  // Liam Refsnes  // Jenny Refsnes  // Anja refsnes |

**Return Values**

Từ khóa void, được sử dụng để xác định rằng phương thức không có giá trị trả về. Nếu muốn phương thức trả về giá trị, sử dụng các kiểu dữ liệu để thay thế (int, char, …), và sử dụng từ khóa return:

|  |  |
| --- | --- |
|  | public class MyClass {  static int myMethod(int x) {  **return** 5 + x;  }  public static void main(String[] args) {  System.out.println(myMethod(3));  }  }  // Outputs 8 (5 + 3) |

# **Java Classes**

## **Java Classes/Objects**

**Java Classes/Objects**

Java là ngôn ngữ lập trình hướng đối tượng.

Mọi thứ trong Java đều được liên kết với các **class** (lớp) và **object** (đối tượng), cùng với **attributes** (thuộc tính) và **methods** (phương thức) của nó.

**Create a Class**

Để tạo class, sử dụng từ khóa class:

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 5;  } |

**Create an Object**

Trong Java, một object được tạo từ một class. Chúng ta đã tạo class có tên MyClass, vậy chúng ta sẽ sử dụng nó để tạo object.

Để tạo một object của MyClass, chỉ rõ tên class, theo sau là tên object và sử dụng từ khóa new:

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 5;  public static void main(String[] args) {  MyClass **myObj** = new MyClass();  System.out.println(myObj.x);  }  } |

**Multiple Objects**

Bạn có thể tạo nhiều đối tượng của một class:

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 5;  public static void main(String[] args) {  MyClass **myObj1** = new MyClass(); // Object 1  MyClass **myObj2** = new MyClass(); // Object 2  System.out.println(myObj1.x);  System.out.println(myObj2.x);  }  } |

## **Java Classes Attributes**

**Java Classes Attributes**

Thuộc tính của một class là các biến của class.

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 5;  int y = 3;  } |

**Accessing Attributes**

Truy cập vào thuộc tính bằng cách tạo một object của class, và sử dụng dấu .

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 5;    public static void main(String[] args) {  MyClass myObj = new MyClass();  System.out.println(myObj.x);  }  } |

**Modify Attributes**

Bạn có thể sửa đổi giá trị của thuộc tính.

|  |  |
| --- | --- |
|  | public class MyClass {  int x = 10;    public static void main(String[] args) {  MyClass myObj = new MyClass();  myObj.x = 25; // x is now 25  System.out.println(myObj.x);  }  } |

Nếu bạn không muốn có thể ghi đè giá trị hiện có của thuộc tính, sử dụng từ khóa final:

|  |  |
| --- | --- |
|  | public class MyClass {  **final** int x = 10;    public static void main(String[] args) {  MyClass myObj = new MyClass();  myObj.x = 25; // will generate an error: cannot assign a value to a **final** variable  System.out.println(myObj.x);  }  } |

## **Java Class Methods**

**Java Class Methods**

Trong phần này chúng ta sẽ nhắc lại về method trong class.

Ví dụ: tạo một method myMethod() trong MyClass:

|  |  |
| --- | --- |
|  | public class MyClass {  static void **myMethod()** {  System.out.println(“Hello World!”);  }  } |

Sử dụng myMethod():

|  |  |
| --- | --- |
|  | public class MyClass {  static void **myMethod()** {  System.out.println(“Hello World!”);  }  public static void main(String[] args) {  **myMethod();**  }  }  // Outputs “Hello World!” |

**Static or Public**

static method: có thể được truy cập ngoài việc phải tạo object

public method: phải tạo object mới có thể truy cập.

Một ví dụ về sự khác nhau giữa phương thức static và public:

|  |  |
| --- | --- |
|  | public class MyClass {  // static method  static void **myStaticMethod()** {  System.out.println(“Static methods can be called without creating objects”);  }  // public method  public void **myPublicMethod()** {  System.out.println(“Public methods must be called by creating objects”);  }  public static void main(String[] args) {  **myStaticMethod();** // call the static method  // myPublicMethod(); This would complie an error    MyClass myObj = new MyClass(); // create an object of MyClass  myObj.myPublicMethod(); // call the public method on the object  }  } |

**Access Methods With an Object**

Ví dụ

|  |  |
| --- | --- |
|  | // create a Car class  public class Car {    // create a fullThrottle() method  public void fullThrottle() {  System.out.println(“The car is going as fast as it can!”);  }  // create a speed() method and add a parameter  public void speed(int maxSpeed) {  System.out.println(“Max speed is: ” + maxSpeed);  }  // inside main, call the methods on the myCar object  public static void main(String[] args) {  Car myCar = new Car(); // create a myCar object    myCar.fullThrolttle(); // call the fullThrottle() method  myCar.speed(200); // call speed() method  }  }  // The car is going as fast as it can!  // Max speed is: 200 |

## **Java Constructors**

**Java Constructors**

Một hàm khởi tạo trong Java là một phương thức đặc biệt được sử dụng để khởi tạo đối tượng. Hàm khởi tạo được gọi khi một object của một class được tạo. Nó có thể được sử dụng để khởi tạo giá trị cho các thuộc tính.

Tạo một constructor:

|  |  |
| --- | --- |
|  | // create a MyClass class  public class MyClass {  int x; // create a class attribute    // create a **class constructor** for the MyClass class  public MyClass() {  x = 5;  }  public static void main(String[] args) {  MyClass myObj = new MyClass(); // create an object of class MyClass (This will **call the constructor**)  System.out.println(myObj.x); // print the value of x  }  }  // Outputs 5 |

**Constructor Parameters**

Hàm khởi tạo có thể có tham số, được sử dụng để khởi tạo giá trị cho thuộc tính.

|  |  |
| --- | --- |
|  | public class MyClass {  int x;    public MyClass(int y) {  x = y;  }  public static void main(String[] args) {  MyClass myObj = new MyClass(5);  System.out.println(myObj.x);  }  }  // Outputs 5 |

Có thể sử dụng nhiều tham số như ý muốn:

|  |  |
| --- | --- |
|  | public class Car {  int modelYear;  String modelName;    public Car(int year, String name) {  modelYear = year;  modelName = name;  }  public static void main(String[] args) {  Car myCar = new Car(1969, “Mustang”);  System.out.println(myCar.modelYear + “ ” + myCar.modelName);  }  }  // Outputs 1969 Mustang |

## **Java Modifiers**

**Modifiers**

Java chia modifier làm 2 loại:

Access Modifiers – kiểm soát mức độ truy cập

Non-Access Modifiers – không kiểm soát mức độ truy cập, nhưng cung cấp các chức năng khác

**Access Modifiers**

Đối với class, có thể sử dụng public hoặc default:

|  |  |  |
| --- | --- | --- |
| Modifier | Description |  |
| public | Class có thể được truy cập bởi một class khác |  |
| *default* | Class chỉ được truy cập bởi các class trong cùng một package (gói). |  |

Đối với **attributes**, **methods** và **constructors**, có các cấp độ truy cập như bên dưới:

|  |  |  |
| --- | --- | --- |
| Modifier | Description |  |
| public | Code có thể được truy cập bởi tất cả các class |  |
| private | Code chỉ được truy cập trong cùng một class |  |
| *default* | Code chỉ được truy cập trong cùng một package. Việc này được sử dụng khi không chỉ định một modifier nào. |  |
| protected | Code có thể được truy cập trong cùng một package và subclasses. |  |

**Non-Access Modifiers**

Đối với class, có thể sử dụng final hoặc abstract:

|  |  |  |
| --- | --- | --- |
| Modifier | Description |  |
| final | Class không thể được kế thừa từ các class khác |  |
| abstract | Class không thể được sử dụng để tạo object. |  |

Đối với attributes và methods, có các cấp độ bên dưới:

|  |  |  |
| --- | --- | --- |
| Modifier | Description |  |
| final | Attribute và method không thể được ghi đè hoặc chỉnh sửa |  |
| static | Attribute và method thuộc về class, chứ không phải là đối tượng |  |
| abstract | Chỉ có thể được sử dụng trong abstract class, và chỉ có thể được sử dụng trong method. Method không có body, ví dụ: abstract void run();. Body sẽ được cung cấp bởi các subclass. |  |
| transient | Attribute và method được bỏ qua khi tuần tự hóa đối tượng của chúng |  |
| synchronized | Method chỉ có thể được truy cập bởi một thread tại một thời điểm |  |
| volatile | Giá trị của một attribute là không được lưu trữ cục bộ theo luồng mà không được đọc từ bộ nhớ chính. |  |

**Final**

Nếu bạn không muốn ghi đè lên các giá trị attribute đã có, hãy sử dụng final:

|  |  |
| --- | --- |
|  | public class MyClass {  **final** int x = 10;  **final** double PI = 3.14;    public static void main(String[] args) {  MyClass myObj = new MyClass();  myObj.x = 50; // will generate an error: can not assign a value to a final variable  myObj.PI = 25; // will generate an error: can not assign a value to a final variable  System.out.println(myObj.x);  }  } |

**Static**

Một phương thức static có nghĩa là có có thể được truy cập ngoài việc tạo object, không giống với public:

Một ví dụ để thấy sự khác nhau giữa static và public:

|  |  |
| --- | --- |
|  | public class MyClass {  // static method  static void **myStaticMethod()** {  System.out.println(“Static methods can be called without creating objects”);  }  // public method  public void **myPublicMethod()** {  System.out.println(“Public methods must be called by creating objects”);  }  public static void main(String[] args) {  **myStaticMethod();** // call the static method  // myPublicMethod(); This would complie an error    MyClass myObj = new MyClass(); // create an object of MyClass  myObj.myPublicMethod(); // call the public method on the object  }  } |

**Abstract**

Một phương thức abstract thuộc về một abstract class, và nó không có body. Body được cung cấp bởi các subclass.

|  |  |
| --- | --- |
|  | // code from filename: Person.java  // abstract class  abstract class Person {  public String fname = “John”;  public int age = 24;  public **abstract** void study(); // abstract method  }  // subclass (inherit from Person)  class Student extends Person {  public int graduationYear = 2019;  public void **study()** {  // the body of the abstract method is provided here  System.out.println(“Studying all day long”);  }  }  // End code from filename: Person.java  // Code from filename: MyClass.java  class MyClass {  public static void main(String[] args) {  // create an object of the Student class (Which inherits attributes and methods from Person)  Student myObj = new Student();    System.out.println(“Name: ” + myObj.fname);  System.out.println(“Age: ” + myObj.age);  System.out.println(“Graduation Year: ” + myObj.graduationYear);  myObj.study(); // call abstract method  }  } |

## **Java Encapsulation**

**Encapsulation**

Ý nghĩa của Encapsulation, là để đảm bảo rằng các thông tin nhạy cảm được ẩn đi đối với người dùng. Để đạt được điều này, bạn phải:

* khai báo variable/attribute như là private (chỉ được truy cập trong cùng class).
* cung cấp setter và getter để truy cập vào và cập nhật giá trị của một private variable.

Get and Set