

# I:Users:a540788:Downloads:java.png Notes

1. Next class October 5th 2015.

# I:Users:a540788:Downloads:java.png Variables, declarations, and coding conventions

## Coding conventions

Oracle has a page for code conventions at <http://www.oracle.com/technetwork/articles/javase/codeconvtoc-136057.html>

## Camel Case

Camel Case is the practice of writing compound words or phrases such that each word or abbreviation begins with a capital letter. Camel case starts with a lowercase letter.

Example:

myFirstVariable

## Primitive Data Types and variable declarations

The Java programming language is statically-typed (all variables need to be declared):

int gear = 1;

the Java programming language supports eight *primitive data types*. A primitive type is predefined by the language and is named by a reserved keyword. Primitive values do not share state with other primitive values:

* **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
* **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
* **int**: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -231 and a maximum value of 231-1. In Java SE 8 and later, you can use the int data type to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1.
* **long**: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -263 and a maximum value of 263-1.
* **float**: The float data type is a single-precision 32-bit IEEE 754 floating point.
* **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. **boolean**: The boolean data type has only two possible values: true and false.
* **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character strings via the [java.lang.String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \t "_blank) class. Enclosing your character string within double quotes will automatically create a new String object; for example:

String s = "this is a string";

String objects are *immutable*, which means that once created, their values cannot be changed. The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such.

# I:Users:a540788:Downloads:java.png Operators and strong typing

## Operators

Java supports wealth of **operators** that allow a program to manipulate values of variables:

### Assignment Operators

|  |  |
| --- | --- |
| **Operator** | **Example** |
| Assignment(**=**) | int a = 100; |
| Addition & assignment (**+=**) | a + = 100; |
| Subtraction & assignment (**−=**) | a − = 100; |
| Multiplication & assignment (**\*=**) | a \* = 100; |
| Division & assignment (**/=**) | a / = 100; |
| Reminder & assignment (**%=**) | a % = 100; |

### Arithmetic Operators

|  |  |
| --- | --- |
| **Operator** | **Example** |
| Addition (**+**) | a = a + b; |
| Subtraction (**−**) | a = a − b; |
| Multiplication (**\***) | a = a \* b; |
| Division (**/**) | a = a / b; |
| Remainder (**%**) | a = a % b; |

### Unary and Equality Operators

|  |  |
| --- | --- |
| **Operator** | **Example** |
| **++** | ++a & a++; increments a by 1 |
| **––** | ––a & a––; decrements a by 1 |
| **!** | !y; negates y |
| **==** | a == b |

### Relational and Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Example** |
| Greater than (**>**) | a > b |
| Greater than or equal to (**>=**) | a >= b |
| Less than (**<**) | a < b |
| Less than or equal to (**<=**) | a <= b |
| Not equal to (**!=**) | a != b |
| Logical AND (**&&**) | (a > b) && (a > c) |
| Logical OR (**||**) | (a > b) || (a > c) |
| Logical NOT (**!**) | complement |
| Ternary (**?:**) | Typical C style if-then-else operator |

## Strong Typing

A strongly typed language has to declare the type of the variable for all variables:

int i = 1;

while a weakly typed language (R for example) would just write:

i = 1;

# I:Users:a540788:Downloads:java.png Conditional flow control

## ifelse

|  |  |
| --- | --- |
| **Syntax** | **Example** |
| if (condition)  {    // true-block of statements  }  else  {    // else-block of statements  } | int y = 0  if (x>10)  {    y = 1;  }  else  {    y = 0;  } |

## Switch

|  |  |
| --- | --- |
| **Syntax** | **Example** |
| switch (expr)  {    case literal-1:      case literal-1 body; break;    case literal-2:      case literal-2 body; break;    case literal-3:      case literal-3 body; break;    default:      default body  } | int monthNumber = 8;  String month;  switch (monthNumber) {  case 1: month = "Jan"; break;  case 2: month = "Feb"; break;  case 3: month = "Mar"; break;  case 4: month = "Apr"; break;  case 5: month = "May"; break;  case 6: month = "Jun"; break;  case 7: month = "Jul"; break;  case 8: month = "Aug"; break;  case 9: month = "Sep"; break;  case 10: month = "Oct"; break;  case 11: month = "Nov"; break;  case 12: month = "Dec"; break;  default: month = "Invalid"; break;  } |

# I:Users:a540788:Downloads:java.png Looping with while, do while, and for loops

## For

|  |  |
| --- | --- |
| **Syntax** | **Example** |
| for(*initialize vars*; *check condition*; *increment/decrement vars*)  {    // block of statements  } | for(int i=1; i<11; i++)  {  System.out.println("Count is: " + i);  } |

## While

|  |  |
| --- | --- |
| **Syntax** | **Example** |
| while(condition)  {    // block of statements  } | int i = 1;  while (i < 11) {  System.out.println("Count is: " + i);  i++;  } |

## Do while

|  |  |
| --- | --- |
| **Syntax** | **Example** |
| do  {    // block of statements  }  while (condition); | int i = 1;  do {  System.out.println("Count is: " + i);  i++;  } while (i < 11); |

# I:Users:a540788:Downloads:java.png Arrays

## One-dimensional arrays

An array is declared as:

String [] names;

double [] measurements;

An example of usage is:

String [] bugs = {"cricket", "beetle", "katydid"};

String [] alias = bugs;

System.out.println(bugs.equals(alias));

System.out.println(bugs.toString());

System.out.println(bugs[0].toString());

Or alternatively:

String [] bugs = new String[3];

Bugs[0] = "cricket";

Bugs[0] = "beetle";

Bugs[0] = "katydid";

String [] alias = bugs;

System.out.println(bugs.equals(alias));

System.out.println(bugs.toString());

System.out.println(bugs[0].toString());

## Loop with an array

A typical loop through an array is:

for (int i = 0; i < myArray.length; i++) {

int item = myArray[i];

sum += item;

}

## Multi-dimensional Arrays

public String[][] bits =

{{ "Michael", "Ernest", "MFE"},

{"Ernest", "Friedman-Hill", "EFH"},

{"Kathi", "Duggan", "KD"},

{"Jeff", "Kellum", "JK"}};

# I:Users:a540788:Downloads:java.png Defining and calling methods

A method usually has this syntax:

modifier returnType nameOfMethod (Parameter List) {

// method body

}

static void action (int number) {

// method body

}

static int anotherAction (float number) {

// method body

}

## What is a package?

A **Java package** is a technique for organizing **Java** classes into namespaces providing modular programming in **Java**. **Java** packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually.

## What is a class?

A **class** is nothing but a blueprint or a template for creating different objects which defines its properties and behaviors. **Java class** objects exhibit the properties and behaviors defined by its **class**. A **class** can contain fields and methods to describe the behavior of an object.

## What is a static method?

**Static methods** in **java** belong to the class (not an instance of it). They use no instance variables and will usually take input from the parameters, perform actions on it, then return some result. Instances**methods** are associated with objects and, as the name implies, can use instance variables.