Cheatsheets / **Java for Programmers**

codecademy

Arrays in Java

Arrays

An array is a collection of values with the same data type.

```
String[] animals = {"Giraffe",
"Elephant", "Toucan"};

// Access an element via its index:
System.out.println(animals[0]); //
Prints: Giraffe

// Change an element value:
animals[1] = "Lion";

// Find number of elements in an array:
System.out.println(animals.length); //
Prints: 3

// Traverse array using for loop:
for (int i = 0; i < animals.length; i++)
{
  System.out.println(animals[i]);
}
/* Prints:
Giraffe
Lion
Toucan
*/

// Traverse array using for-each loop
for (int i: animals) {
  System.out.println(i);
}
/* Prints:
Giraffe
Lion
Toucan
*/
```

2D Arrays

A 2D array is an array that stores arrays of the same data type.

```
// Declare a 2D array that stores char arrays
char[][] letters = {{'A', 'a'}, {'B', 'x'}, {'C', 'c'}};

// Access an element via its index:
System.out.println(letters[0][1]); // Prints: a

// Change an element by accessing its index:
letters[1][1] = "b";

// Iterate over 2D array in row-major order:
for (int i = 0; i < letters.length; i++){
  for (int j = 0; j < letters[0].length; j++){
    System.out.print(letters[i][j]);
  }
}
// Prints: AaBbCc

// Iterate over 2D array in column-major order:
for (int i = 0; i < letters[0].length; i++){
  for (int j = 0; j < letters.length; j++){
    System.out.print(letters[j][i]);
  }
}
// Prints: ABCabc
```

Print Share

Methods

A method is a modular, reusable block of code that can be called throughout a program to complete a certain task.

```
/*
The following method is a public method called findSum. The method takes in two int parameters called int1 and int2. This method returns an int value.
*/
public static int findSum(int num1, int num2) {
  return num1 + num2;
}

public static void main(String[] args) {
  // Call the method with the arguments 3 and 4
  int sum = findSum(3,4);
  System.out.println(sum); // Prints: 7
}
```

Variable Types

Variables are used to name, store, and reference different types of data.
Primitive data types are predefined types of data and include `int`, `double`, `boolean`, and `char`.
Reference data types contain references to an object. An example reference data type is `String`.

```
// int - stores whole numbers:
int num = 10;

// double - stores decimal numbers:
double dec = 4.99;

// boolean - stores true or false values:
boolean isTrue = true;

// char - stores a single character value:
char firstLetter = 'A';

// String - stores multiple characters:
String message = "hello there";
```

Conditional Statements

In Java, conditional statements execute code based on the truth value of given `boolean` expressions.

```
boolean expression1 = false;
boolean expression2 = false;
boolean expression3 = true;

if (expression1) {
  System.out.println("The first
expression is true");
} else if (expression2) {
  System.out.println("The second
expression is true");
} else if (expression3) {
  System.out.println("The third
expression is true");
} else {
  System.out.println("All other
expressions were false");
}
// Prints: The third expression is true
```

Comparison and Logical Operators

Conditional operators and logical operators evaluate the relationship between values in order to determine a `true` or `false` value.

```
// Comparison Operators:
int a = 1;
int b = 5;
System.out.println(a > b); // Prints:
false
System.out.println(a < b); // Prints:
true
System.out.println(a >= 1); // Prints:
true
System.out.println(a + 4 <= b); //
Prints: true
System.out.println(a == 1); // Prints:
true
System.out.println(b != 5); // Prints:
false

// Logical Operators:
System.out.println(!true); // Prints:
false
System.out.println(!false); // Prints:
true

System.out.println(true && true); //
Prints: true
System.out.println(true && false); //
Prints: false
System.out.println(false && true); //
Prints: false
System.out.println(false && false); //
Prints: false

System.out.println(true || true); //
Prints: true
System.out.println(true || false); //
Prints: true
System.out.println(false || true); //
Prints: true
System.out.println(false || false); //
Prints: false
```

String Methods

Java's `String` class has many useful methods including:

- `.length()`, which returns the length of the `String`
- `.concat()`, which concatenates two `String`s together
- `.equals()`, which checks for `String` equality
- `.indexOf()`, which returns the index of the first occurrence of a specified character
- `.charAt()`, which returns the character at a specified index
- `.substring()`, which extracts a substring

```
// Using the .length() method:
String str = "Hello World!";
System.out.println(str.length()); //
Prints: 12


// Using the .concat() method:
String name = "Code";
name = name.concat("cademy");
System.out.println(name); // Prints:
Codecademy


// Using the .equals() method:
String flavor1 = "Mango";
String flavor2 = "Matcha";
System.out.println(flavor1.equals(flavor2
)); // Prints: false


// Using the .indexOf() method:
String letters = "ABCDEFGHIJKLMN";
System.out.println(letters.indexOf("C"));
// Prints: 2


// Using the .charAt() method:
String currency = "Yen";
System.out.println(currency.charAt(2));
// Prints: n


// Using the .substring() method
String line = "It was the best of times,
it was the worst of times.";
System.out.println(line.substring(26));
// Prints: it was the worst of times.
System.out.println(line.substring(7,
24)); // Prints: the best of times
```

Loops

Java has four kinds of loops that rely on a `boolean` condition and continue to iterate until the condition is no longer true:

- `while` loops
- `do-while` loops
- `for` loops
- `for-each` loops

```
// An example of a while loop:
int x = 0;
while (x < 2) {
  System.out.println(x);
  x++;
} // Prints: 0 and 1
```

```
// An example of a do-while loop:
do {
  System.out.println("Impossible!");
} while (2 == 4); // Prints: Impossible!
```

```
// An example of a for loop:
for (int i = 0; i < 10; i++) {
        System.out.println(i);
} // Prints: 0 to 9, inclusive
```

```
// An example of a for-each loop:
String[] colors = {"Red", "Blue",
"Yellow"};
for (String c : colors) {
        System.out.println(c);
} // Prints: Red, Blue, and Yellow
```

break and continue

Java has two keywords that help further control the number of iterations in a loop:

- `break` is used to exit, or break, a loop. Once `break` is executed, the loop will stop iterating.
- `continue` can be placed inside of a loop if we want to skip an iteration. If `continue` is executed, the current loop iteration will immediately end, and the next iteration will begin.

```
// An example of a break statement:
for (int i = 0; i < 10; i++) {
  System.out.println(i);
  if (i == 4) {
    break;
  }
} // Prints: 0 to 4, inclusive
```

```
// An example of a continue statement:
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < numbers.length; i++)
{
  if (numbers[i] % 2 == 0) {
    continue;
  }
  System.out.println(numbers[i]);
} // Prints 1, 3, and 5
```

Print Share

—

static attributes

static <type> <varName>

Declaring Variables

static byte age = 30;

—

instance attributes

<type> <varName>

Declaring Variables

byte age = 30;

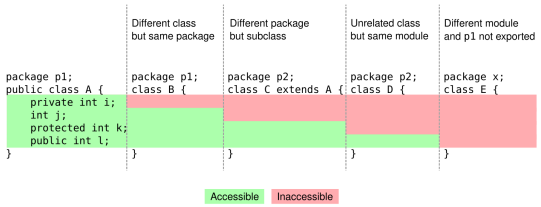
long viewsCount = 3123456L;

float price = 10.99F;

char letter = ‘A’;

boolean isEligible = true;

access



Simple main

```
public class Main  {

  public static void main(String[] args)  {

  }

}
```

Output

System.out.println("Hello World!");
System.out.print("Hello World!");

System.out.printf("Hello World!"); //this can run format แบบในไพท่อน เช่น

System.out.printf("Test %d Test %s", 5, “this Test”);

//output is : Test 5 Test this Test

\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

String.format ใช้แบบปริ้น f แต่จะไม่ปริ้น แต่รวมไว้เป็นสตริง

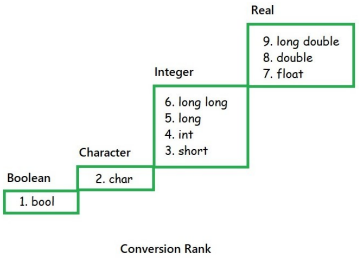
Java Type Casting

To convert a string to a number, we use one of the following methods:

• Byte.parseByte(“1”)
• Short.parseShort(“1”)
• Integer.parseInt(“1”)
• Long.parseLong(“1”)
• Float.parseFloat(“1.1”)
• Double.parseDouble(“1.1”)

java implicit casting

ตัวอย่าง



long = 100000l;

int var;

u can do this : var = (int) long;

Reading Input

Scanner scanner = new Scanner(system.in);

double number = scanner.nextDouble();
byte number = scanner.nextByte();
String name = scanner.next();
String line = scanner.nextLine();
.close(); ระวังลืม

Comparison Operators

== , !=, <, >, <=, >=

Logical Operators

• x && y (AND): if both x and y are true, the result will be true.

• x || y (OR): if either x or y or both are true, the result will be true.

• !x (NOT): reverses a boolean value. True becomes false.

Array Class

Methods ที่น่าใช้

asList() Returns a fixed-size list backed by the specified Arrays

compare(array 1, array 2) Compares two arrays passed as parameters lexicographically.

copyOf(originalArray, newLength) Copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.

copyOfRange(originalArray, fromIndex, endIndex) Copies the specified range of the specified array into a new Arrays.

equals(array1, array2) Checks if both the arrays are equal or not.

fill(originalArray, fillValue) Assigns this fill value to each index of this arrays.

setAll(originalArray, functionalGenerator) Sets all the elements of the specified array using the generator function provided.

sort(originalArray) Sorts the complete array in ascending order.

sort(originalArray, fromIndex, endIndex) Sorts the specified range of array in ascending order.

sort(T[] a, Comparator< super T> c) Sorts the specified array of objects according to the order induced by the specified comparator.

Java ArrayList Methods

ArrayList<type> var = new ArrayList<type>();

ArrayList<Class> var = new ArrayList<Class>();
สามารถใส่ Class ใน ArrayList ได้

KEY WORD

package
import java.util.*;

- ArrayList
- Collections
- Comparator
- Arrays
- Scanner

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;

enum **<name>** {}

class <name> extends <Superclass> implements class1, class2
interface class1{} !ระวังถ้าประกาศตัวแปรในนี้จะเป็น final
interface class2{} !ระวังถ้าประกาศตัวแปรในนี้จะเป็น final
abstract class subclass จะต้องมี con ที่กำหนด

toString
super.
this.

import java.util.Comparator;
จากห้องเรียน

```
public class FavoriteCourseComputer implements
Comparator<Student23> {

   @Override

   public int compare(Student23 o1, Student23 o2) {

      return
o1.favoriteCourse.compareTo(o2.favoriteCourse) ;

   }

}
```

Exception Handling

```
try {

} catch (ExceptionType e1) {

} catch (Exception e2) {

} finally {}
```

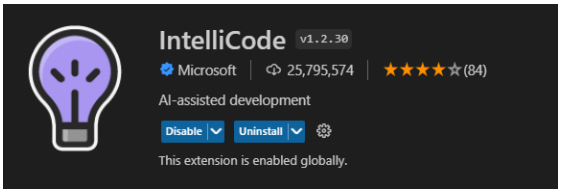
Arrays key word

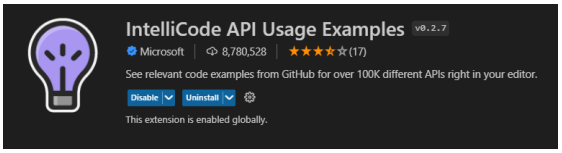
- <type> [] <varName>;
- <type> [] <varName> = {“val”, “val, “val”, “val”};
- int x = myValues [1][2];
- int[][]myValues = { {1,2,3,4}, {5,6,7}};
- **.clone();**
- **.length; !no()**
- .equals(another array); !return bool
- Arrays.toString(array); !return no void

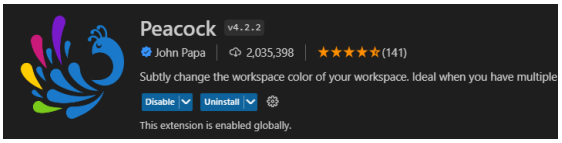
Arrays key word

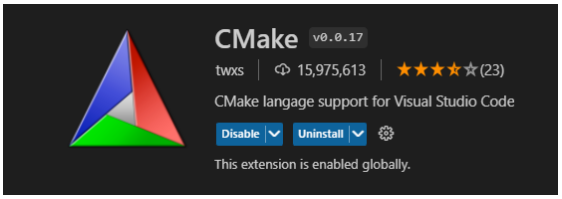
- <> เขียนด้วยถ้าสีแดง
- `Arraylist<type> name= new Arraylist<type>();`
- `name.add(val);`
- `name.set(<index>,val);`
- `name.size();`
- `name.toString();`

แนะนำงับ









Array vs ArrayList

Array	ArrayList
length predefined	size() variable
manual shift	Auto shift
faster	slower
primitive+Ref type	ref type only
equals() not overriden	overriden equals()
toString() not overriden	overriden toString()
Arrays.sort()	Collections.sort()
Arrays.binarySearch()	Collections.binarySearch()

Java String Methods

s.length()	length of s
s.charAt(i)	extract ith character
s.substring(start, end)	substring from start to end-1
s.toUpperCase()	returns copy of s in ALL CAPS
s.toLowerCase()	returns copy of s in lowercase
s.indexOf(x)	index of first occurence of x
s.replace(old, new)	search and replace
s.split(regex)	splits string into tokens
s.trim()	trims surrounding whitespace
s.equals(s2)	true if s equals s2
s.compareTo(s2)	0 if equal/+ if s > s2/- if s < s2

JAVA - ArrayList - Cheat Sheet

`ArrayList<String>listName = new ArrayList<String>();`	*Declaring an ArrayList*
`ArrayList<String> listName = new ArrayList<String>(5);`	*Declaring an ArrayList with specific index size (5)*
`listName.add("penguin");`	*Adding to ArrayList*
`listName.remove(0);`	*//removes index [0]*
`listName.remove("penguin");`	*//removes string penguin wherever it is*
`listName.set(0, "tux");`	*Replacing an existing Item in ArrayList*
`listName.size();`	*Checking the Size (how many indexes)*
`listName.indexOf(item)`	*Searching under which index is (item)?*
`int index? = myArrayList.indexOf("penguin");`	
`listName.contains(item);`	*Verifying Contents*
`if( myArrayList.contains("penguin") )`	*(is there an item with such and such name or value)*
`listName.isEmpty();`	*Checking if Empty*
`while( myArrayList.isEmpty() );`	
`newListName.addAll(listName);`	*copy the contents of an existing ArrayList to the new one.*
`ArrayList<String> copyArrayList = new ArrayList<String>();`	
`copyArrayList.addAll(myArrayList);`	
`listName.clear();`	*Clearing an ArrayList*
`Collections.sort(listName);`	*Sorting an ArrayList*
`for( <type> varName : listName )`	*Outputing an ArrayList*
`System.out.println(varName);`	
`for( String ix : myArrayList )`	
`System.out.println(ix);`	
`for(Object ix : myArrayList )`	
`System.out.println(ix);`	
`listName.toArray(arrayName);`	*Conversion - ArrayList to an Array*
`String[] regArray = new String[myArrayList.size()];`	
`myArrayList.toArray(regArray);`	
`ArrayList listName = Array.asList(arrayName)`	*Array to an ArrayList*
`ArrayList<String> myArrayList = Arrays.asList(regArray);`	