

Cheatsheet Java

Comments

Single-line Comment:
1 String txt = "Hello!";
2 //this is a Comment
3 System.out.println(txt);

Multi-line Comment:
1 String txt = "Hello!";
2 /*Comments will not be
3 executed */
4 System.out.println(txt);

Control structures

1 if(condition1){
2 /*if condition1 true,
3 execute*/
4 }
5 else if(condition2){
6 /*if condition1 false and
7 condition2 true, execute */
8 }
9 else{
10 //if everything false, execute
11 }

Loops

1 for(int i=0; i<10; i++){
2 //execute 10 times
3 }
4 while(condition){
5 //execute as long as condition
6 }
7 //Do-While Loop
8 do{
9 //execute at least once
10 }while(condition);
11 //For-Each Loop
12 for(List<String> s : list1){
13 s.length();
14 }

Switch

1 switch(expression){
2 case 1:
3 //execute if expression==1
4 break;
5 case 2:
6 //execute if expression==2
7 break;
8 default:
9 //execute if expression is
10 not 1 or 2 */
11 break;
12 }

Functions

1 //Delaration and Implementation
2 <ret-type> <func-name>(<para-type>
3 <para-name>, ...){
4 // function body
5 //execute
6 return <expression>;
7 }
8 //Function call
9 <func-name>(<argument>, ...);

Operations

Arithmetic:

Operation	Example
+	3 + 5 == 8
-	7 - 2 == 5
*	4 * 2 == 8
/	7 / 2 == 3
% (Modulo)	72 % 10 == 2

Comparison:

Operator	Math	Example
>	>	5 > 2
>=	≥	5 >= 2
<	<	10 < 21
<=	≤	5 <= 5
==	=	5 == 5
!=	≠	-32 != 32

Types

Primitive data types:

Type	Size	Type	Size
byte	8 bit	float	32 bit
short	16 bit	double	64 bit
int	32 bit	Type	Value
long	64 bit	char	'a', 'G'
		boolean	true, false
		void	-

Typecasting: byte → short → char → int → long → float → double

Non-Primitive data types:

Type	Value
String	"Hello World!"
Array	int[] myNum = {10, 20, 30, 40};

Declaration, Initialisation

Declaration: int a; String txt;
<Type>< Name>;

Initialisation: int b = 50; int b = a;
<Type><Name>=<Literal/Variable>;

Assignment: a = b; txt = "abc";

Arrays

1 //Declaration
2 <type>[] <name>;
3 int[] arr;
4 //allocation
5 <name> = new <type>[<size>];
6 arr = new int[5];
7 //or
8 <name> = {<element1>, ...};
9 arr = {1, 2, 3, 4, 5};
10 //Access
11 <name>[<index>];
12 arr[2] = 5;

Strings

1 /*Strings are immutable and come
2 with a number of methods
3 already implemented*/
4 //Declaration
5 String <name>=new String(<value>);
6 String helloString=new String("hello");
7 //or
8 String <name>=<value>;
9 String helloString="hello";
10 //Small Selection of useful Methods
11 helloString.length();
12 helloString.charAt(<index>);
13 helloString.split(" ");

Collections

Common datatypes are implemented in the java.util package:

1 import java.util.*;
2 /* Lists are ordered collections of
3 objects, similar to arrays */
4 List<type><name>=new ArrayList<>();
5 List<String> list1=new
6 ArrayList<>();
7 list1.add("Hello");
8 list1.add("World");
9 System.out.println(list1.get(1));
10 /*Sets are unordered and duplicate
11 free collections of objects */
12 Set <type><name> = new HashSet<>();
13 Set<String> set1 = new HashSet<>();
14 set1.add("1");
15 set1.add("2");
16 set1.add("1"); //not added
17 System.out.println(set1);
18 //Output either [1, 2] or [2, 1]
19 //Maps let you access data via a
20 key
21 Map<type1, type2><name> = new
22 HashMap<>();
23 Map<Integer, String> map = new
24 HashMap<>();
25 map.put(23, "foo");
26 map.put(28, "foo");
27 map.put(23, "bar"); //overwrites 23
28 System.out.println(map);
29 //Output {23=bar, 28=foo}
30 System.out.println(map.get(23));
31 //Output 'bar'

Object-Oriented Programming

Attributes:
Data (Variables)
Describes the State of the Object
Modifier always private

Methods:
Code/Function
Changes the state of the object,
Or interacts with other objects
Modifier mostly public

1 // Defining Class
2 class <class-name>{
3 //Attributes
4 <modifier> <type> <var-name>;
5 //Methods
6 <modifier> <ret-type> <func-name>
7 >(<para-type> <para-name>,
8 ...){
9 // function body
10 }
11 }
12
13 class Room {
14 private int chairs = 4; // Attribute
15
16 public void addChairs(int chairs)
17 {
18 this.chairs += chairs;
19 } //Method
20 }
21
22 //Creating Object
23 class <name> <obj-name> =
24 new <class-name>();
25 Room kitchen = new Room();
26
27 //Accessing Attributes and Methods
28 <obj-name>.<var-name>; //Attribute
29 kitchen.chairs;
30
31 <obj-name>.<func-name>
32 (<argument>, ...); //Method
33 kitchen.addChairs(2);
34
35 /*to access members of own class
36 use keyword this:*/
37 this.<var-name>;
38 this.<func-name>(<argument>, ...);
39 this.chairs += 5;
40
41 Access modifiers to define access to an
42 attribute or method:
43 • public: Anyone can access the member,
44 default
45
46 • private: Only the class itself can access
47 the member
48
49 • protected: Only the class itself and its
50 subclasses can access the member
51
52 Constructor:
53 same name as class
54 will get called if a new object is created
55 mostly used for Initialisation of attributes
56
57 1 class <class-name> {
58 2 public <class-name>(...){
59 3 //constructor body
60 4 }
61 5 ...
62 6 }
63 7 class Student {
64 8 public Student(String name, ...){
65 9 this.name = name;
66 10 ...
67 11 }
68 12 }

Inheritance

1 /* To give a subclass all members
2 of a superclass
3 to inherit use 'extends' keyword */
4 class Vehicle {
5 ...
6 }
7 class Car extends Vehicle {
8 ...
9 }
10
11 /* use 'super' to refer
12 to the superclass */
13 class <Subclass-name> extends
14 <Superclass-name> {
15 public <Subclass-name>(...){
16 super();
17 }
18 }
19
20 /*use @Override to replace a
21 method from the superclass */
22 @Override
23 public <Superclass-Method>(){
24 /* calls the method
25 of the superclass */
26 super.<Superclass-Method>();
27 //insert own code here
28 }

Abstract Classes and Inheritance

1 /* Abstract classes cannot be
2 instantiated and need to be
3 inherited by subclasses,
4 abstract functions are declarations
5 of functions that have to be
6 implemented in subclasses */
7 public abstract class <class-name>
8 {
9 //abstract method
10 public abstract <ret-type> <func-
11 name>(...);
12 ...
13 }
14
15 /* Interface is a group of
16 related methods with no
17 implementation. A class can
18 implement multiple interfaces */
19 public interface <interface-name> {
20 public <ret-type> <func-name>
21 (...);
22 }
23
24 public class <class-name>
25 implements <interface-name> {
26 ...
27 }
28 }

Static Variables, Static Functions

Static variables are variables that
can be accessed from every object
of the class. Only one copy of the
variable exists. Static Functions are
Functions with one implementation
for every object of the class. Cannot
access instance variables or meth-
ods directly. Can be accessed via
Class name
1 public class Test{
2 public static int counter;
3
4 public static int getCounter(){
5 return counter;
6 }
7 }
8
9 //getCounter() can be accessed via
10 Test.getCounter();

Generics

Generics are used to create classes
for different data types:
1 public class Tuple<T> {
2 private T item1, item2;
3 public void set(T item1, T
4 item2) {
5 this.item1 = item1; // ...
6 }
7 public T get(int index) {
8 //return item1 or item2
9 }
10 }