Cheatsheet Java

Comments

Single-line Comment:

```
String txt = "Hello!";
//this is a Comment
System.out.println(txt);
```

Multi-line Comment:

```
String txt = "Hello!";
/*Comments will not be
executed */
System.out.println(txt);
```

Control structures

```
if(condition1){
  /*if condition1 true,
  execute*/
}
else if(condition2){
  /*if condition1 false and
  condition2 true, execute */
}
else{
  //if everything false, execute
}
```

Loops

```
for(int i=0; i<10; i++){
  //execute 10 times
}
while(condition){
  //execute as long as condition
}
//Do-While Loop
do{
  //execute at least once
}while(condition);
//For-Each Loop
for(List<String> s : list1){
  s.length();
}
```

Switch

```
switch(expression){
  case 1:
    //execute if expression==1
    break;
  case 2:
    //execute if expression==2
    break;
  default:
    /*execute if expression is
     not 1 or 2 */
    break;
}
```

Functions

```
//Delaration and Implementation
<ret-type> <func-name>(<para-type>
      <para-name>, ...){
    // function body
    //execute
    return <expression>;
}
//Function call
<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
>=	$\geq$	5 >= 2
<	$<$	10 < 21
<=	$\leq$	5 <= 5
==	$=$	5 == 5
!=	$\neq$	-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: $byte \rightarrow short \rightarrow char \rightarrow int \rightarrow long \rightarrow float \rightarrow double$

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

```
//Declaration
<type>[] <name>;
int[] arr;
//allocation
<name> = new <type>[<size>];
arr = new int[5];
//or
<name> = {<element1>, ...};
arr = {1, 2, 3, 4, 5};
//Access
<name>[<index>];
arr[2] = 5;
```

Strings

```
/*Strings are immutable and come
with a number of methods
already implemented*/
//Declaration
String <name>=new String(<value>);
String helloString=new String("
      hello");
//or
String <name>=<value>;
String helloString="hello";
//Small Selection of useful Methods
helloString.length();
helloString.charAt(<index>);
helloString.split(" ");
```

Collections

Common datatypes are implemented in the java.util package:

```
import java.util.*;
/* Lists are ordered collections of
objects, similar to arrays */
List<type><name>=new ArrayList<>();
List<String> list1=new
ArrayList<>();
list1.add("Hello");
list1.add("World");
System.out.println(list1.get(1));
/*Sets are unordered and duplicate
free collections of objects */
Set <type><name> = new HashSet<>();
Set<String> set1 = new HashSet<>();
set1.add("1");
set1.add("2");
set1.add("1"); //not added
System.out.println(set1);
//Output either [1, 2] or [2, 1]
//Maps let you access data via a
      key
Map<type1, type2><name> = new
      HashMap<>();
Map<Integer, String> map = new
      HashMap<>();
map.put(23, "foo");
map.put(28, "foo");
map.put(23, "bar"); //overwrites 23
System.out.println(map);
//Output {23=bar, 28=foo}
System.out.println(map.get(23));
//Output 'bar'
```

Object-Oriented Programming

Attributes:

define the state of an Object

Data

Describes the Object

Other names: fields, properties

Modifier always private

Methods:

describes behavior of an Object

Code/Function

Changes the state of the object

Or interacts with other objects

Modifier mostly public

```
// Defining Class
class <class-name>{
  //Attributes
  <modifier> <type> <var-name>;
  //Methods
  <modifier> <ret-type> <func-name
      >(<para-type> <para-name>,
      ...){
    // function body
  }
}
class Room {
  private int chairs = 4; //
      Attribute
  public void addChairs(int chairs)
  {
    this.chairs += chairs;
  } //Method
}

//Creating Object
<class-name> <obj-name> =
new <class-name>();
Room kitchen = new Room();

//Accessing Attributes and Methods
<obj-name>.<var-name>; //Attribute
kitchen.chairs;

<obj-name>.<func-name>
(<argument>, ...); //Method
kitchen.addChairs(2);

/*to access members of own class
      use keyword this:*/
this.<var-name>;
this.<func-name>(<argument>, ...);
this.chairs += 5;
```

Access modifiers to define access to an attribute or method:

- public: Anyone can access the member, default
- private: Only the class itself can access the member
- protected: Only the class itself and its subclasses can access the member

Constructor:

same name as class

will get called if a new object is created

mostly used for Initialisation of attributes

```
class <class-name> {
  public <class-name>(...){
    //constructor body
  }
  ...
}
class Student {
  public Student(String name, ...){
    this.name = name;
    ...
  }
}
```

Inheritance

```
/* To give a subclass all members
of a superclass
to inherit use 'extends' keyword */
class Vehicle {
  ...
}
class Car extends Vehicle {
  ...
}


/* use 'super' to refer
to the superclass */
class <Subclass-name> extends
<Superclass-name> {
  public <Subclass-name>(...){
    super();
  }

  /*use @Override to replace a
  method from the superclass */
  @Override
  public <Superclass-Method>(){
    /* calls the method
    of the superclass */
    super.<Superclass-Method>();
    //insert own code here
  }
}
```

Abstract Classes and Inheritance

```
/* Abstract classes cannot be
instantiated and need to be
inherited by subclasses,
abstract functions are declarations
of functions that have to be
implemented in subclasses */
public abstract class <class-name>
      {
  //abstract method
  public abstract <ret-type> <func-
      name>(...);
  ...
}


/* Interface is a group of
related methods with no
implementation. A class can
implement multiple interfaces */
public interface <interface-name> {
  public <ret-type> <func-name>
  (...);
}

public class <class-name>
      implements <interface-name> {
  ...
}
```

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 methods directly. Can be accessed via Class name

```
public class Test{
  public static int counter;

  public static int getCounter(){
    return counter;
  }
}

//getCounter() can be acccessed via
Test.getCounter();
```

Generics

Generics are used to create classes for different data types:

```
public class Tuple<T> {
  private T item1, item2;
  public void set(T item1, T
    item2) {
      this.item1 = item1;// ...
  }
  public T get(int index) {
      //return item1 or item2
  }
}
```

Official Documentation: https://docs.oracle.com/en/java/javase/18/docs/api/index.html
Educational: https://www.w3schools.com/java/