Java Mastery - Part 2 Cheat Sheet

by Bayan (Bayan.A) via cheatography.com/122738/cs/22936/

if then else

if statement identifies the statement/code block to run based on the expressions value(the specific condition)

Code block is defined by curly braces	{can contain 1 or more statements}

else comes after the if, executed when the condition is `false`

else if can test multiple conditions

Example

```
int score = 6000;
if (score >= 5000) {
  do something;
} else if (score <1000 && score >= 500) {
  do something else;
] else [
  do this if all previous conditions are false
}
```

Method

Prevent code duplication

Same as 'function' in other languages

Set the name of the method, then the data type and name of parameters it will access from the `main` method

Java automatically creates variables with the appropriate data type for us

Defining a new method:

```
public static void method Nam e(b oolean boolParam, int intPar am) {
}
```

`void` means to "return nothing"

Calling a Method

When the method is called, all the code in curly brackets { } in the method is executed

Execute by **calling** the method (in `main`) with the **required arguments** in the brackets

Arguments can be the variable name or the actual values we want to send, as long as it matches the parameters

```
method Nam e(v ari able1, variab le2);
```

In the example, two arguments expected

A variable can be assigned a method result

e.g.

```
int highScore = calcul ate Sco re( gam eOver, score, level, bo nus);
```

When the variable highScore is printed, the result of the calculations is sent back from the calculateScore method and assigned ot the variable

Return variable from method to main:

```
public static int method Nam e(b oolean boolParam, int intPar am) {
}
```

Changing `void` to `int` means we are returning an `int` data type into the Main Method

The method must explicitly return he variable:

```
return newValue;
```

An `int` variable is expected, whether or not the statements in the method execute (i.e. when an if statement is either true or false, something must be returned)

Return variable from method to main: (cont)

All the { } in the program must be accounted f

Full example:

`public static void main(S tring[]`	otherwise an error.
`boolean isThis Helpful = true;`	
`int points = 15;`	
`method Nam e(i sTh isH elpful, po`	datatype need not be specified
`}`	

```
public static int method Nam e(b
 oolean isThis Hel pful, int poi
nts) {

 if (isThi sHe lpful) {
  int finalScore = points + 5
  System.ou t.p rin tln ("Your f
inal score is: " + finalS core);
  return finalS core;
  }
return -1;
```

//-1 indicates invalid value/not found

Method Overloading

Allows us to create multiple methods with the s **different parameters**

Changing the data type of the method will **not r** signature. **The number of paramaters make it u**

The methods can have the same name but **diff**

```
sum(int a, int b);
sum(int a, int b, int c);
sum(int a, int b, int c, int d);
```

```
public static int sum(int a, int b
return a + b;
 }
```

```
public static int sum(int a, int b
return a + b + c;
 }
```

```
public static int sum(int a, int b
return a + b + c + d;
 }
```

pintln methid is an example of method overloa methods of the same name

Method Overloading (cont)

It improves code readability and re-usability

Easier to remember one method name isntead of many

Achieves naming consistency

Gives flexibility to call a similar method with different types of data, based on defined arguments/parameters

Switch

Similar to the if-then-else statement

Good for testing values of the same variable

The variable that will be changed goes inside the brackets

the conditions tested will be the **case** numbers

break; are essential to close off your case comparision. Without it, results will be unpredictable.

The final switch statement is **default**, same as **else**

Switch statement

```
int value = 1;
switch (value) {
case 1:
System.ou t.p rin tln ("Value was 1");
break;
case 2:
System.ou t.p rin tln ("Value was 2");
```

Switch (cont)

```
break;
default:
System.ou t.p rin tln ("Was not 1 or 2");
break;
}
```

Case tests can also be on one line, e.g.:

```
int value = 1;
```

Setting the variable to be changed

```
if (value == 1) {
System.ou t.p rin tln ("Value was 1");
} else if (value == 2) {
System.ou t.p rin tln ("Value was 2");
} else {
System.ou t.p rin tln ("Was not 1 or 2");
}
```

Structure:

```
for(init; termination; increment)
{
    case 3:case 4:case 5:
}
```

Common Methods

variable.**toLowerCase()**	Turns the string variable to all lower case
variable.**toUpperCase()**	Turns the string variable to all UPPERCASE
Math.round(variable)	Rounds decimal numbers to the nearest value
String.format("%.2f", variable)	Converts and outputs the variable number with just two decimale points
Math.sqrt(variable);	Variable must be long. If not, add (long) before the expression
variable1.equals(variable2)	Tests if one String variable is equal to another

For Loop/Statement

Processes a part of a code block until a condition is satisfied

Variable created in the for statement only exists in that code block

init:	Code **initialised** once at the start of the loop
termination:	Determines at what point it exits the for loop
	Once it evaluates to false, it will exit the loop and proceed to the next line
increment	An expression invoked after each iteration

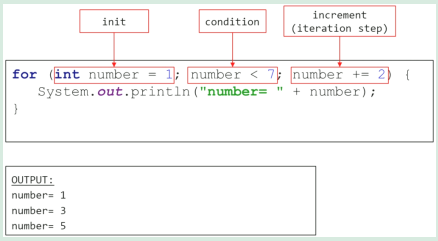
Example:

```
for(int i = 0; i < 5; i++)
```

int i = 0	initialise i to zero,
i < 5	test if i is less than 5 and keeps processing until i is greater than 5
i++	add 1 to the value of i

Looping forwards or backwards depends on the conditions and ranges

For Statement



while/do-while

Similar to the for loop

instead of looping a certian number of times(as seen in `for`), `while() allows you to loop until the expression is true or false

the count must be incremented, otherwise you will enter an infinite loop

for version

```
for(int i = 0; i != 5; i++) {
//do something }
```

while version

```
int i = 1
while(i != 6)
{
//do something }
```

while

while(condition {
//statements
}

```
while( true) {
if(i == 5) {
break;
}
count++;
```

Do-While

It will always execute once or more depending on defined expressions

do {
//statements
} while(condition);

Remember, semi-colon after while condition

```
count = 1;
 do {
 count++;
 } while( count != 6);
```

Object-Oriented Programming

Object	A value of a class type
	Has states and behaviours
	State is stored in fields
	e.g. a dogs name, breed, colour
	Behaviour is shown via methods
	e.g. a dogs behaviour: barking, wagging tail, running
Class	Describes set of objects with the same behaviour
	Can have any number of methods to access the value of different methods

Updating a variable using a method instead of directly

Why?	If the variables access modifier is set to `private` for security reasons, we don't want users to directly change the variable by making it 'public'

Use `this.` before the class to refer to a particular field

Object-Oriented Programming (cont)

`this.model = model;`	this means to update the field model with the contents of the parameter `model` that was passed through.

Classes can have:
Local variables: Defined in methods, constructors or blocks
Instance variables: variables withina class but outside any method. Initialised when the class in instantiated.
Class variables: Variables declared within a class, outside any method, with the static keyword

Static Variables

Declared using the keyword `static`

Also known as **static member variables**

Every class instance shares the same static variable	Chang made that variab will be seen i other instan

Accessible by **static methods** directly

Reading user input with **Scanner**, `scanner` is declared as a static variable

e.g. `private static String name;`	not a good idea

Instance variables

They **don't** use the static keyword

Known as fields or member variables

Belong to an **instance** of a class

Represents the state of an isntance

Instance variables (cont)

Every instance:	-Has it's own copy of an instance variable -Can have a different value(-state)

Static Method

Declared using a static modifier

Can't access instance methods and instance variables directly

Cannot use the `this` keyword	Usually used for operations that don't require data from an instance of the class (from `this`)

When a method **does not use instance variables**, that method should be **declared as a static method**

e.g. main is a static method and is called by JVM when it starts an application

Example

```
public static void printS um(int
 a, int b) {
 System.ou t.p rin tln ("sum = "
 + (a + B));
}
```

Instance Methods

Belongs to an instance of a class

To use, we must instantiate the class first by using the new keyword

Can access instance methods and variables directly

Can access static methods and static variables directly

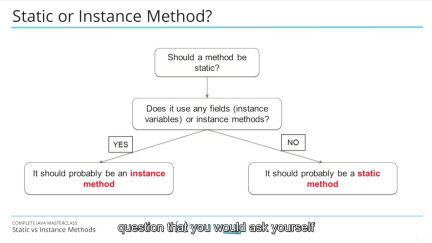
Example:

```
class Dog {
 public void bark() {
 System.ou t.p rin tln ("wo of")
;
 }
}
```

Instance Methods (cont)

```
public class Main {
 public static void main(S tring [] args)
 {
       Dog rex = new Dog(); //create insta
nce
  `rex.b ark(); //call instance method`
 }
}
```

Static or Instance Method?



Should a method be static?
|
V
Doies it use any fields(instance variables) or Instance methods?

YES? It should probably be an instance method

NO? It should probably be a static method