Loops

`while` Creates a loop with condition for continuing.

`do` Creates a loop like `while`, but whose condition is evaluated after the iteration, not before.

`for` Creates a loop with counter initialization, condition for continuing, and counter increment.

`continue` Stops the execution of innermost loop, and continues with next iteration. With label: continues next iteration of labeled loop.

`break` Stops and jumps out of innermost loop, continuing with subsequent code. With label: jumps out of labeled loop.

```
for(int i = 0, i < 10, i++)
{
    System.out.println("i: " + i);
}
```

Exceptions

`throw` Throws an exception.

`throws` Indicates that a method can throw one or multiple exceptions.

`try` Opens a block for intercepting exceptions.

`catch` Opens a block for handling exceptions that occurred in the `try` block.

`finally` Opens a block that is always executed after the `try` (and `catch`).

```
FileReader reader = null;
try
{
    reader = new FileReader(myFile);
    ...
}
catch (Exception e)
{
    e.printStackTrace();
}
finally
{
    reader.close();
}
```

Control flow

`if` Executes block if Boolean condition is true.

`else` Executes block if previous conditions are false.

`switch` Executes a block of code depending on the value of a specified variable (byte/Byte, short/Short, int/Integer, char/Character, String).

`case` Defines a `case` label for a particular value in a `switch` block.

`default` Defines a `default` label in a `switch` block, to be executed from if none of the `case` labels match the value of the variable in question.

`break` Jumps out of innermost loop or `switch` statement. With label: jumps out of labeled loop/`switch`.

`assert` Verifies that a condition is satisfied; otherwise, throws an error with the specified message.

`instanceof` Tests whether an object is an instance of the specified type, or one of its subtypes.

`return` Stops the execution of a method, and (if a `return` type is specified) send a value back to the original caller of the method.

`synchronized` While `synchronized` methods or blocks are being executed on an object, only one thread can access the object at a time.

JAVA

The keywords of the language

abstract	else	interface	switch
assert	enum	long	synchronized
boolean	extends	native	this
break	*false***	new	throw
byte	final	*null***	throws
case	finally	package	transient
catch	float	private	*true***
char	for	protected	try
class	*goto**	public	void
*const**	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	
double	int	super	

(*) *reserved but not used*

(**) *reserved but not keyword*

Types and objects

`class` Defines a class type.

`interface` Defines an interface type, which specifies behavior without implementing it.

`enum` Defines an enum type.

`extends` Indicates that a class inherits from another class, or that an interface derives from other interfaces.

`implements` Indicates that a class implements the methods specified in one or multiple interfaces.

`import` Gives direct access to specific types, to all the types in a package, or to static methods, without needing to use their fully qualified names.

`this` Reference to the current object, in non-static contexts.

`super` Reference to the superclass of the current object.

`package` Indicates the package to which the type belongs.

`abstract` Indicates that a class or method is abstract (must be implemented via inheritance).

`native` Indicates that a method is implemented in native code, in a language other than Java, and in another file.

Access modifiers

`private` Indicates that a member is only accessible from inside the class where it is defined.

package-private (default) Indicates that a type or member is accessible from everywhere in the package, but not from outside the package.

`protected` Indicates that a member is accessible from everywhere in the package, and also from all subclasses of the member's class.

`public` Indicates that a type or member is accessible from everywhere.

Miscellaneous

*`false`*** Boolean literal: represents truth value of a condition that does not accurately describe the data.

*`true`*** Boolean literal: represents truth value of a condition that accurately describes the data.

`new` Operator that instantiates an object.

*`null`*** Special literal that represents the value of the null reference.

`transient` Marks a variable as not serializable.

`static` Indicates that a variable, method, or block belongs not to an instance, but to the class itself.

`strictfp` Guarantees that all floating-point calculations in a method or class will produce the same results on all machines, conforming to the IEEE 754 specification.

`final` Indicates that a class, method, or field cannot be extended or modified.

`volatile` Guarantees the synchronization of a variable in a multi-thread context, ensuring that value changes are always visible to other threads.

```
public enum Sex
{
    FEMALE("f"), MALE("m");
    private final String code;

    Sex(final String code)
    {
        this.code = code;
    }

    public String getCode()
    {
        return code;
    }

    public static Sex valueOfCode(String code)
    {
        for ( Sex s : values() )
            if ( s.code.equals(code) )
                return s;
        throw new IllegalArgumentException();
    }
}
```

```
public interface Animal
{
    String getCall();
}

public class Wolf implements Animal
{
    protected String call;

    public Wolf()
    {
        call = "howl";
    }

    @Override
    public String getCall()
    {
        return call;
    }
}

public class Dog extends Wolf
{
    public Dog()
    {
        call = "bork";
    }
}
```

Primitive types

`void` Indicates that a method returns no value.

`boolean`	Boolean	8 bits (2 values)
`char`	Character	16 bits
`byte`	Integer	8 bits signed
`short`	Integer	16 bits signed
`int`	Integer	32 bits signed
`long`	Integer	64 bits signed
`float`	Floating-point	32 bits signed
`double`	Floating-point	64 bits signed