General Computer Science I

Java Cheatsheet

Maximilian Kratz, Christopher Katins, Jannis Bluml ¨

Version: April 14, 2020 at 6:34 p.m

Notice

This cheatsheet summarizes most of the basics of the Java programming language, but is by no means all-encompassing. It was compiled by tutors over the past few years and is largely based on the revision course on Java from the software internship course from the winter semester 2017/2018.

Furthermore, we would like to point out once again that this document is not permitted for the exam!

Properties of Java

• Class-based and object-oriented programming. • Type safe. • High level of awareness
(most commonly used programming language). • Operating system independence. • Simple error/exception
handling. • Wide range of libraries and tools. • Simple syntax.

data types

The type	Meaning	Operators	examples
	Primitive data types up to 231 ÿ		
int	Integers from ÿ2 31	1 + - / % / + - / + -	2, 3, -5, 21221213
float	Floating point numbers up to 32 bits		2.3, 3.4, -1.3
double	Floating point numbers up to 64 bits		2.3, 3.4, -1.3
char	single character/character value		’A’, ’1’, ’%’
boolean	truth values	&& | |	true, false
	(More complex) data types string of		
String	characters + (concatenation) ”Hello”, ”2.5”, ”CB”		
Self-created objects			An object of class Graph has type Graph.

Common errors in Java

1. Java is case sensitive, i.e. upper and lower case letters play a role.
2. For every opening bracket there must also be a matching closing bracket.
3. Don't forget semicolons. 4. Static

variables/methods and non-static variables. Only static methods can be used in static methods variables are used.

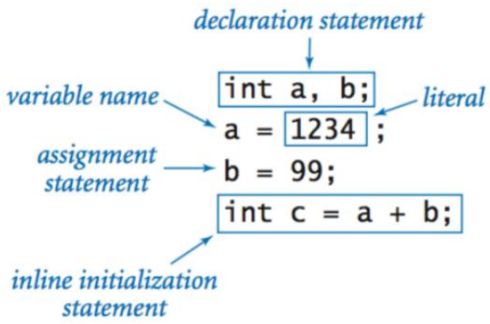
5. Do not confuse =, == and equals() .
6. The index starts at 0 not at 1.
7. If exceptions are expected/thrown, they must be handled. For example with try catch-Bl¨ocken.
8. Does not modify lists while iterating over them.
9. Don't forget to check handovers (e.g. for zero).

Operators

Operators – Overview –

sign meaning		Comment
Arithmetic operators and assignment		
=	allocation	
+	Addition	
-	Subtraction	
*	multiplication	
/	Division	No division by 0
%	Module	remainder of a division
increment and decrement		
++	Increment by 1	
--	Decrement by 1	
a += b Corresponds to a=a+b *=, /=, ...		
See +=		
Logical Operators		
<, >	Smaller bigger	
<=, >= Less than, ...		
==	Same	Beware of not primitive data types see ==
!=	unequal	
&&	And	
||	Or	

Operators - Example -




```
int a = 5; int b
= 7; boolean c
= a < b; // Result of a logical
operator // is always a logical value.
```

```
int a = 5; int b
= a++; // b = 5 and a=6 b += ++a; // b = 12
and a = 7 // Warning a++ and ++a do //
different things.
```

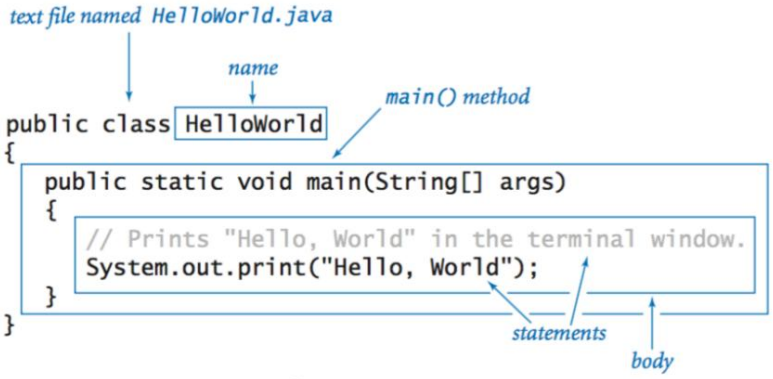
Operatorpr¨azedenz

++, –	increment and decrement
+, -	Unary plus and minus (sign)
	bitwise complement
	logical complement
(Type)	Cast
*, /, %	Multiplikation, Division, Rest (Modulo)
+, -	addition and subtraction
+	String-Konkatenation
<<	shift left
>>	Right shift with sign extension
>>>	Right shift without sign extension
<, <=, >, >= Numerical comparisons	
instanceof	type comparison
==, !=	Equality/inequality of values/references
&	And
	Free
	Or
| &&	Logical conditional And logical
|| ?:	conditional Or conditional operator
	(ternary operator)
=	Assignment
+=, &=, *=, ... Assignment with operation	

Note: We have not met and used all of these operators.

class structure

Hello World



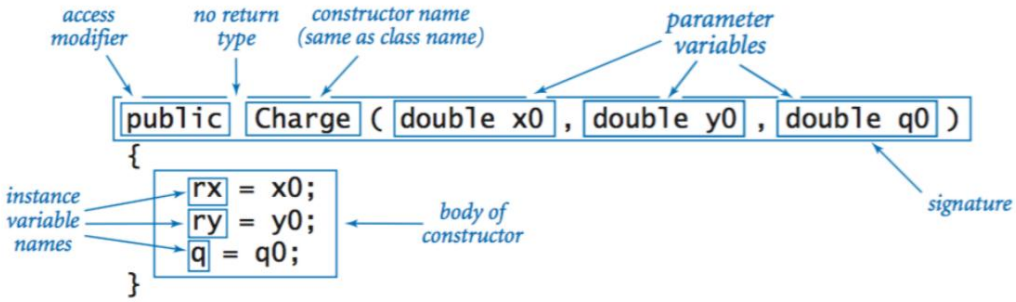
Classes always consist of: • Access

specification (here public). • Class name, corresponding

to the file name (here HelloWorld). • The keyword Next can have a class: • Object variables. • Methods.
• Constructor. main" method. class.

•

Constructor



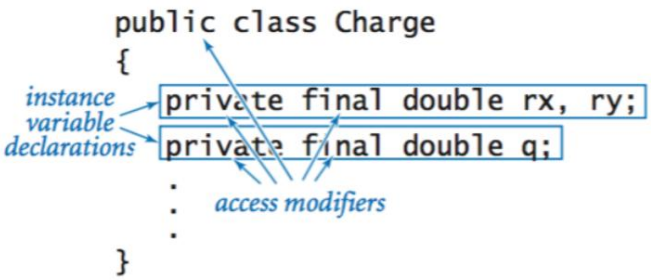
A constructor consists of: • Access

specification (here public). • No return value (no void

either). • Class name (here batch). • Transfer parameter(s) (can

also be empty ÿ (), here x0, y0, q0).

variables



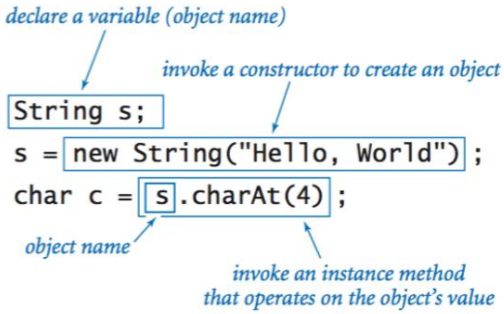
A variable consists of: • Access specification (here private, final). • Variable name.

Note: A variable can already be initialized when it is declared.

Note: Several variables can be declared at the same time.

Note: Variables can be created globally (instance variable) or locally.

objects



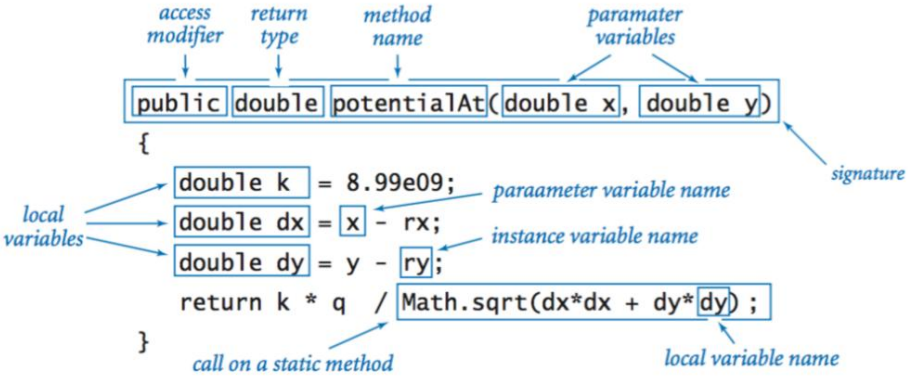
An object consists of: • Its type (class, here String). • A name (here s).

Note: An object is initialized using the new keyword .

Note: An object is initialized using its constructor.

Note: With instance_name." one accesses the methods of the instance.

methods



A method consists of: • Access

specification (here public). • Return type (if no return type,

then void, here double). • Method name (here potentialAt). • Transfer parameter(s) (can also be empty ÿ (),

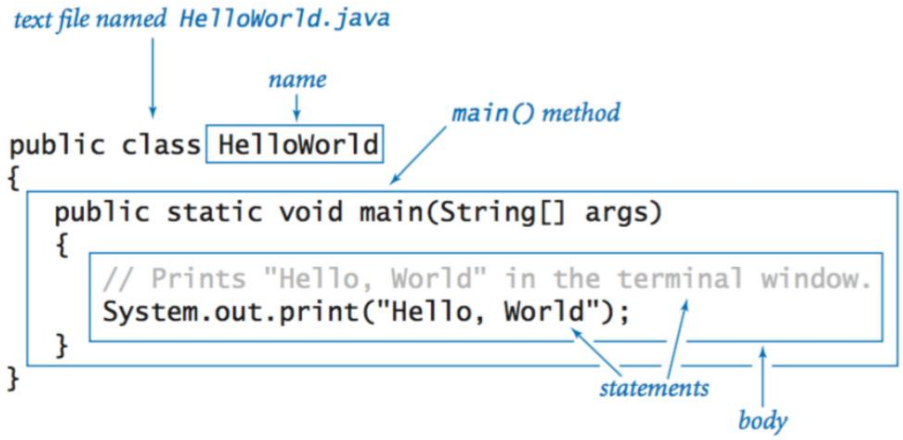
here x,y). • Method body (started with { and ended with }).

Note: A method with a return type always requires a return statement.

Note: A method is invoked by its name.

Note: Even if there are no transfer parameters, the brackets (()) are required.

Main



A main method has the following structure:

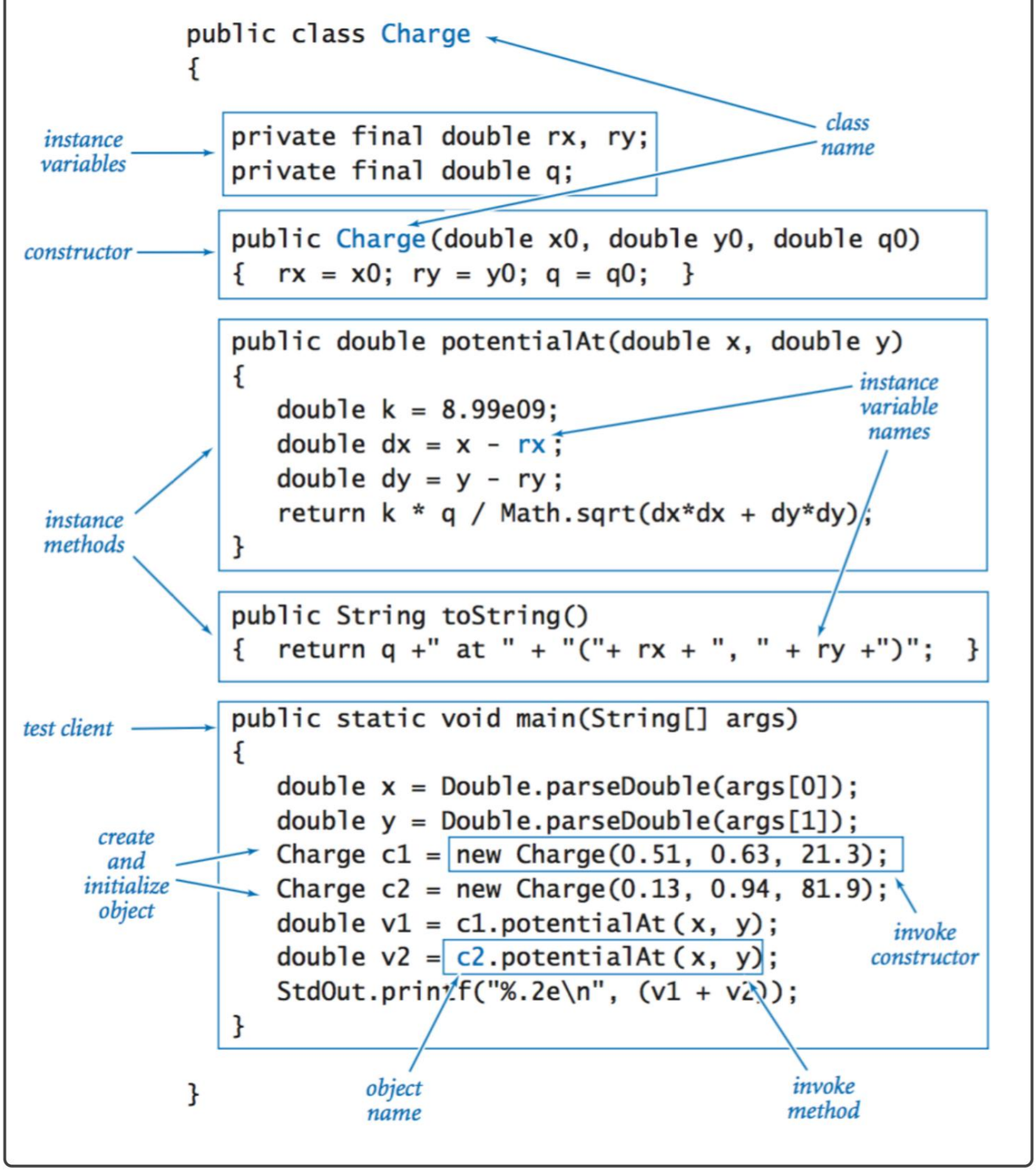
• Has public static as the access specification. • The return type is void. •

The method name is main. • The method is

given an array called ”args” of type String .

Note: The main method is executed when the file is exported.

example class



modifiers

access modifiers

• public: Always accessible. •
protected: Accessible only from the package and its heirs. • No modifier
(also called “package-private”) (default): Accessible from the package. • private: Accessible only from your own class.

Not access modifiers

• static: Generates class methods/variables. The classes do not haveto be initialized for this. • final: Methods, variables, etc. declared with final can no longer be used after initialization
to be changed. A declaration without initialization is not possible.
• abstract: Creates abstract methods and classes that cannot be initialized.

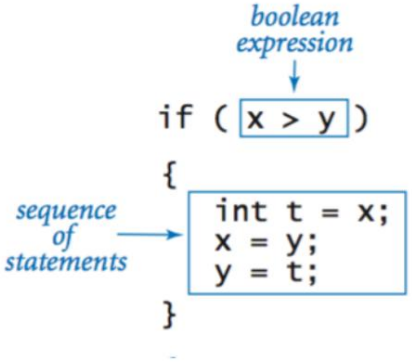
control structures

if-else condition

An if condition consists of: • The keyword • A condition if.
(here x > y). This must become a true evaluate value. • A
body (denoted by { and }). • An if condition can be ver
will see. • Can
be with. - If (x > y) then a, else b”.

Hint: Executes the body if the condition is met.

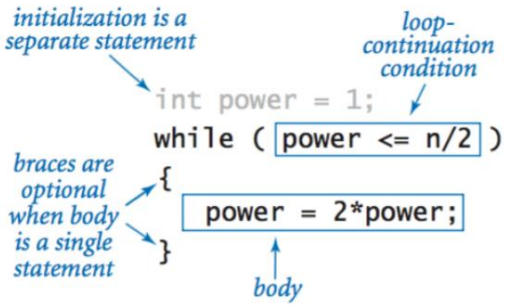
if-else condition - Example -




```
if(x > y) { //...
            a: if x>y } else { //... b:
otherwise

}
```

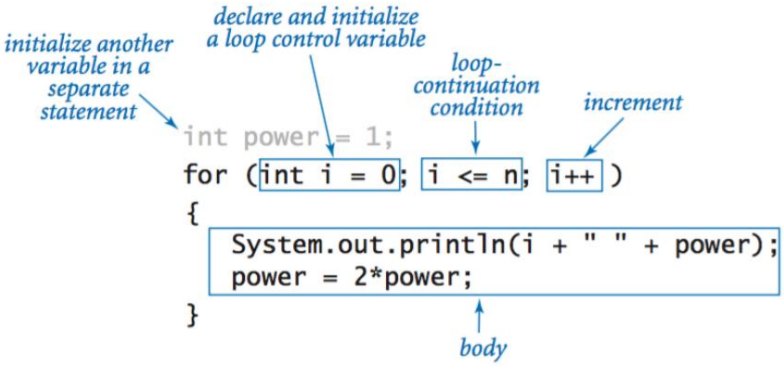
while loop



A while loop consists of: • The keyword
while. • A termination condition (here
power <= n/2). This must evaluate to a truth value. • A loop body (denoted by { and }).

Hint: Repeats the loop body until the condition is no longer fulfilled.

for loop



A for loop consists of: • The keyword •
A loop variable (here i), this for.
can also be declared and initialized at this point. • A termination condition (here i <= n). This must evaluate to a truth value. • An increment or decrement of the loop variable. • A loop body (denoted by { and }).

foreach loop

Example:
for(int i: list){ //... }

A foreach loop consists of: • The keyword •
A loop variable (here i), this for.
must be of the same type as the elements in the list • A loop body (identified by { and }).

be cunning.

Hint: Calls the body for all elements within the list.

Break

```
int factor;
for (factor = 2; factor <= n/factor; factor++)
   if (n % factor == 0) break;

if (factor > n/factor)
   System.out.println(n + " is prime");
```

A break statement interrupts the current control structure. For example, within a loop, this is terminated by a break .

Switch Case

```
switch (day) {
   case 0: System.out.println("Sun"); break;
   case 1: System.out.println("Mon"); break;
   case 2: System.out.println("Tue"); break;
   case 3: System.out.println("Wed"); break;
   case 4: System.out.println("Thu"); break;
   case 5: System.out.println("Fri"); break;
   case 6: System.out.println("Sat"); break;
}
```

A case distinction using a variable (here day).

try-catch-Blöcke

The try-catch statement encloses a section of code and is used to catch m within that section Possible errors (Exceptions) of code so that you can act on it. The general syntax is shown below:

```
try {
        // code that runs safely } catch
(ExceptionClassName variableName) {
        // error handling
}
```

It consists of four parts. The block enclosed by try runs safely. This means that exceptions that may be thrown in this block are caught by the program. Error handling takes place within the catch block. The exception class name describes the error to which the program wants to react. The variable name names the exception within the catch block, so that the exception within the catch block can be reacted to accordingly.

If the caught exception is thrown within the try block, code execution jumps directly to the catch block. If the exception isn't thrown, the catch block is never called.

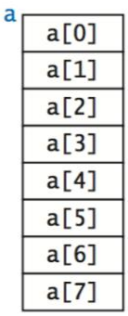
The finally block is simply appended to the end of the try-catch blocks . Its semantics are simple: the finally block is always executed, regardless of whether an exception was thrown in the try block or not:

```
try {
        // code that runs safely } catch
(ExceptionClassName variableName) {
        // error handling } finally
{ // is always
        executed
}
```

The finally block is particularly useful when an original state is to be restored or cleaned up. So it is suitable z. B. very good to close files again.

data structures

Array



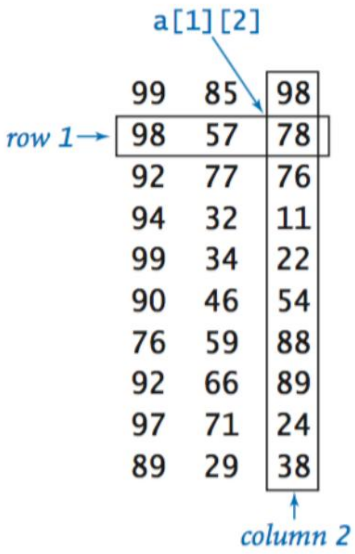
array initialization

```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };

String[] RANKS = {
   "2", "3", "4", "5", "6", "7", "8", "9", "10",
   "Jack", "Queen", "King", "Ace"
};
```

Collection of elements of one type. Arrays have a fixed size that cannot be changed after initialization.

2D Arrays



2D array initialization

```
double [][] a =
{
   { 99.0, 85.0, 98.0,  0.0 },
   { 98.0, 57.0, 79.0,  0.0 },
   { 92.0, 77.0, 74.0,  0.0 },
   { 94.0, 62.0, 81.0,  0.0 },
   { 99.0, 94.0, 92.0,  0.0 },
   { 80.0, 76.5, 67.0,  0.0 },
   { 76.0, 58.5, 90.5,  0.0 },
   { 92.0, 66.0, 91.0,  0.0 },
   { 97.0, 70.5, 66.5,  0.0 },
   { 89.0, 89.5, 81.0,  0.0 },
   {  0.0,  0.0,  0.0,  0.0 }
};
```

Arrays can also be assigned during initialization. Two-dimensional arrays are no exception.

Lists

A list is a collection that indexes its elements. This allows adding, modifying and removing elements by an integer index. A list can store elements of any more complex type. Primitive data types cannot be used directly. However, there are wrapper classes for this, such as Integer, which can be used instead of int . Lists also allow nulls and duplicates.

Beispiel:

```
List<String> listStrings = new ArrayList<String>(); listStrings.add("One");
listStrings.add("Two");
listStrings.add("Three");
listStrings.add("Four");
```

Sets

A set that cannot contain duplicates. (For all pairs of elements in the set, e1.equals(e2) is false .) • add, which adds a value to the set . ¨ • contains, which returns true if the passed element is
contained in the set . • remove, which removes the passed value, if any. •
size, toArray, clear, ...

Maps

A map is an object that stores a value for certain keys. This is comparable to a table that has two columns. A map cannot contain duplicates in the keys. Each key refers to exactly one value. Similar to List, Map can only deal with more complex data types, but not with primitive data types. Java has the basic functions via the map interface: • put, which adds a key-value pair to the map . ¨ • get, which determines the value for a given key. • remove, which removes a key-value pair. • containsKey, containsValue, size, empty, ...

Example:

```
public static void main(String[] args) {
        HashMap map = new HashMap();

        // Three objects of class Student_in are created.
        Student_in st1 = new Student_in("Pot", "Hans", "12345"); Student_in st2 = new Student_in("Teller", "Hannes",
        "12323"); Student_in st3 = new Student_in("Cutlery", "Maxi", "12345");


        // Insert the objects into the HashMap.
        // Matriculation number is entered as key. map.put(st1.get matriculation number(),
        st1); map.put(st2.get matriculation number(), st2);


        // Student_in st1 is replaced by st3 because the // matriculation number is already assigned
        as a key. map.put(st3.get matriculation number(), st3);

}
```

The convention

naming conventions

	First letter example		particularities
Class	Large	Charge	Mostly noun
Variable	Klein	numOfBatteries	
Method Small		potentialAt(x,y) Often begins with a verb, possibly	
Constant Uppercase only PI			with underscores: NEW_PI
Package	Lowercase only utility		

Note: Java distinguishes between upper and lower case letters.

Note: Multi-word names in Batteries").

- CamelCase" (each word starts with a capital letter, e.g.
- numOf

class structure

A class is structured as follows:

1. Initial comment on the class. 2. package and import statements.
3. Class declaration/header (= definition of the class!).
4. Variables and constructors (static variables, instance variables, constructors).
5. Methods (grouped by functionality).

The declaration

The following should be noted for declarations: • One declaration per line. • If possible, also initialize variables where they are declared. • Always try to make declarations in blocks (eg at the beginning of a method). • Methods are separated by a blank line.

Statements

The following should be noted for statements/commands:

- One statement per line (try to have no more than one semicolon per line except in exceptions such as in for loops).
- The return statements only have parentheses if they are used for operator precedence. • Control structures always have a body with brackets, the opening bracket is on the same line like the definition.

Example:

```
public int faculty(int k) {
        int f = 1; // Each statement on its own line.

        if(k > 0){ // { On the same line as the if condition.
                f = f * k; k--; //
                No more than a statement, like "f = f * k; k--;".
        }

        return f; // No unnecessary parentheses, like (f).
}
```

Comments

Block Comments

```
/*
  * Is a comment.
  */
```

Single-Line Comments

```
...
/* is a comment. */
...
```

End-of-Line Comments

```
... // Is a comment.
```

JavaDoc

• Documentation comments are written in HTML. • They start with / ÿ ÿ. • They consist of two parts.

– Descriptive text.

– Tags.

• Each class and method should have a Javadoc comment. • Frequently used tags:

– @param, @author, @return, @see, @throws, @since, @deprecated, @version, @date, ...

Documentation (Doc) Comment Example

```
/** *
Returns an Image object that can then be painted on the screen.
* The url argument must specify an absolute {@link URL}. The name * argument is a specifier that is
relative to the url argument. * <p> * This method always returns immediately, whether or not the
* image
exists. When this applet attempts to draw the image on * the screen, the data will be loaded.
The graphics primitives * that draw the image will incrementally paint on the screen.


*
* @param url an absolute URL giving the base location of the image * @param name the location of the
image, relative to the url argument * @return
                        the image at the specified URL
* @see                  Image
*/
public Image getImage(URL url, String name) {
        try
            { return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
                return null;
        }
}
```

inheritance

Casting

• Casting a primitive type changes the type itself and changes the value irreversibly. • Casting with more complex types/objects does not change the type or the object itself. • Casting changes the reference to the object, ie the object is just relabeled, so to speak. • Not every cast is possible (see ClassCastException)!

instanceof Operator

```
// Example: if
(animal instanceof Cat) { ((Cat)
    animal).meow();
}
```

• The instanceof operator evaluates to a boolean value.

• Checks the type of an object against the passed type.

ClassCastException

```
// Example:
Animal animal;
String s = (String) animal;
```

erroneous • At Casts becomes one ClassCastException thrown,

Upcasting

Beispiel:

```
public class Animal { public
    void eat() { // ...

    }
}

public class Cat extends Animal { public void
    eat() { // ...

    }

    public void meow() { // ...

    }
}

Cat cat = new Cat();
Animal animal = cat;
```

Features: •
Casting from a subclass to a superclass. • Generalizes the class. • Subclass specific methods are not ver manoeuvrable.

• Is rarely used.

Downcasting

Beispiel:

```
public class Animal { public
    void eat() { // ...

    }
}

public class Cat extends Animal { public void
    eat() { // ...

    }

    public void meow() { // ...

    }
}

Animal animal = new Cat();

if(animal instanceof Cat)
    ((Cat) animal).meow();
```

Features: •
Casting from a superclass to a subclass. • Specifies the class. • Extends the super class with specific metho the.

• For secure downcasting, instanceof is used beforehand .

inheritance

General Information: • In inheritance,
the base or parent class is also known as the superclass or superclass. • Child classes, that is, those that inherit from others, are considered subclasses or subclasses of the parent class
designated.
• Children inherit all information and methods from the parent class. • Each class inherits from the Object class by default. • The super keyword is used to access the parent class. • final methods cannot be overridden. ¨

Example:

```
class Animal {

    public Tier() { } void

    moves()
        { System.out.println("Swim, run, crawl, hop or fly?");
    }
}
class Vogel extends Tier {

    public Vogel(){ super();

    }
    @Override
    void moves()
        { System.out.println("I'm flying");
    }
    void tweet()
        { System.out.println("tirilli");
    }
    void frisst()
        { System.out.println(getClass().toString() +                    " eats");
    }
}
class Wurm extends Tier { //...


    @Override
    void moves()
        { System.out.println("I'm crawling");
    }
    void frisst()
        { System.out.println(getClass().toString() +                    " eats");
    }
}
```

Note that: • Birds and worms
inherit from animals (keyword extends). • All three classes implement the movesSich() method . • Bird and worm override the method of the superclass in this case. • Overridden methods should be marked with the @Override annotation . The compiler learns
by specifying a method of the parent class.
• The parent class is accessed with super() .

Abstract methods/classes

• An abstract method is a method that is only declared but not defined (implemented). • It is marked with the modifier abstract and with a semicolon after the declaration completed.
• A class that contains an abstract method must itself be marked with abstract . • Such a class cannot be instantiated, ie there is no constructor and no instantiation with new.
• Any subclass of an abstract class that does not implement all abstract methods (through overwrite) must itself be abstract.
• Abstract methods cannot be static, private , or final . • A class without abstract methods can also be declared with abstract (then it cannot be instantiated become!).

interfaces

methods of an interface interface must precede the definition (instead of class) . • The keyword • All are (implicitly) public, ie private and protected are forbidden. • All methods of an interface are (implicitly) abstract. • An interface may only have attributes that are static and final . • An interface is not instantiable and must not have a constructor. • When a class implements an interface, this is indicated by the implements keyword . • The class must implement all methods or be abstract.

Miscellaneous

output

The simplest output is via the command line/console using one of the following commands: System.out.print("One output"); System.out.println("An output followed by a line break");

this

The keyword this can be used to access your own instance. Thus we can access the instance variables via this .

Example:

```
public class Point { public int
        x = 0; public int y = 0;


        //constructor public
        Point(int x, int y) { this.x = x; this.y = y;


        }
}
```

Tests

JUnit

• To be able to use JUnit, junit.jar must be available as a compile-time library. • Terms: – A test class is a class that contains methods for
testing code. The only condition is that it must be instantiable by a public default constructor. Test methods are marked in the test class using the JUnit annotations.

– Test methods denote methods that are marked as such by annotations. Any method that has the visibility attribute public , requires no parameters, and returns void as the return type may be marked as a test method.

– A test case is first of all a specific procedure how a piece of software is to be tested, it includes test values and the expected result. Depending on the complexity of the functionality to be tested, test cases of the same type can be combined into one test method. • Usage: Test methods are denoted by the JUnit @Test annotation . The test framework provides the class org.junit.Assert for checking . If a mismatch occurs, a java.lang.AssertionError or an error derived from it is thrown.

• Defining the initial state: It often happens that many or even all test cases of a test class require an identical environment. To avoid having to write these preparations down again in each test method, JUnit offers the @Before annotation.

• Annotations:

Annotation	Description
@Test	Identification as a test case.
@Test(expected = Exception.class) Marking as a test case with the specification that the test only is successful if the requested exception occurs.	
@Test(timeout = ...ms)	Identification as a test case with the definition that the test is successful if the required time in ms is not exceeded.
@Before	Execution before each call of a test method to set up a defined test environment.
@After	Execution after each call of a test method to do cleanup work.
@BeforeClass	A method marked in this way must be statically defined. This annotation is used to mark a one-time execution before all other test methods are called.
@AfterClass	A method marked in this way must be statically defined. This annotation is used to mark a one-time execution after all other test methods have been called.
@Ignore(Comment)	Method is temporarily not executed. It is imperative that a comment is included in the transfer; this is output in the log of the test run.

Example:

```
public class CollectionTest {
    @Test
    public void sortedSet() { Set<String> s =
        new TreeSet<String>(); s.add("B¨ar"); s.add("Aal");
        Iterator<String> iter
        = s.iterator();
        assertEquals("Aal", iter.next()); assertEquals("B¨ar",
        iter.next());

    }
}
```