Cheatography

OOv1 Cheat Sheet

by Philip Schmid (Higarigh) via cheatography.com/20304/cs/3111/

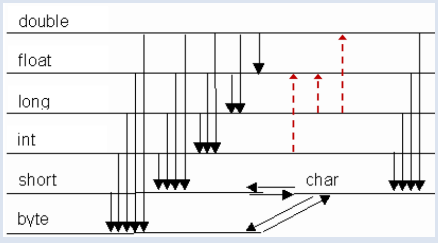
Java basics

`x = a ? b : c`	a true? x = b, a false? x = c
`0.1 + 0.1 == 0.3`	False, workaround: `Math.abs(0.1 + 0.1) < 0.2e6`
`a && b`	Only check b if a is true
`a || b`	Only check b if a is false
`0b11001`	binary, leading "0b"
`0x1e`	hexadecimal, leading "0x"
`010 != 10`	Leading 0 means octal
`'A' == 'a'`	False (case sensitive)
`'A' < 'B'`	True

Java primitive data types

boolean	true, false
char	16 bit, UTF-16
byte	8 bit, -128...127
short	16 bit, -32.768 ... 32.767
int	32 bit, -2^{31} to $+2^{31}-1$
long	64 bit, -2^{63} to $+2^{63}-1$, `long x = 100l;`
float	32 bit, `float x = 100f;`
double	64 bit, `double x = 100d;`

Java type casting



red arrows =implizit (probably information loss due inaccurate dataformat)
black arrows = explizit cast (heavy information loss possible --> developer)

Java interfaces

Methods in intefaces are implicitly public and abstract and can't be private.

Only constant variables are allowed: `public static final int HIGHWAY_MIN_SPEED = 60;` "public static final" is optional.

Same named methods in basic interface and super interface must have the same return type.

Same named variables in basic interface and super interface can have different return types.

Default methods: Basic interface doesn't have to implement default methods. If one method is not overriden, it just takes the default method.

Java reference types

`int[] x = new int[10];`	normal Array (`public int[] x(int[] y) {return z}`)
`int[][] m = new int[2][3];`	2 dimensional array
`Arrays.equals(a, b)`	Compare array content
`Arrays.deepEquals(a, b)`	Compare x dimensional array
`public enum Weekday{ MONDAY, ..., SUNDAY }`	Enum
`String a = "Prog1";`	new reference to String-Object "Prog1" (if already exists, otherwise create it)

Java reference types (cont)

`String c = new String("Prog1");`	new String-Object
`a.equals(b)`	Compare Strings

```
StringBuilderbuilder = new StringBuilder("Hi");
builder.append(" you!");
builder.insert(0, "XY"); String text= builder.toString();
```

Java equals() example

```
@Override
public boolean equals(Object obj) {
  if (obj == null) {
    return false;
  } else if (getClass() != obj.getClass()) {
    return false;
  } else if (!super.equals(obj)) {
    return false;
  } else {
    Student other = (Student)obj;
return regNumber == other.regNumber;
  }
}
```

If equals() is changed, hashCode() has also be changed! x.equals(y) -> x.hashCode() == y.hashCode()

Keywords

public	can be seen by all that imports this package
protected	can be seen by all classes in this package and all subclasses of this

By **Philip Schmid** (Higarigh)
cheatography.com/higarigh/

Published 25th January, 2015.
Last updated 15th January, 2015.

Cheatography

OOv1 Cheat Sheet

by Philip Schmid (Higarigh) via cheatography.com/20304/cs/3111/

Keywords (cont)

package	can be seen by all classes in this package
private	can be seen only by this class
static	only once for all instances of this class
final	can only be defined once and not changed later. Class: no subclasses, Method: no overriding
static final	Means this is a constant

Javadoc

Start with /**	end with */	each line *
@author name	author	class / interface
@version number	version	class / interface
@param name description	parameter	method
@return description	returnvalue	method
@throws/@exception type description	potential exception	method
@deprecated description	deprecated (outdated)	method

Java hashcode() example

```
public int hashCode() {
    return firstName.hashCode() +
31 * surName.hashCode();
}
```

Java compareTo example

```
class Person implements
Comparable<Person> {
  private String firstName,
lastName;
  // Constructor…
  @Override
  public int compareTo(Person
other) {
     int c =
compareStrings(lastName,
other.lastName);
     if (c != 0) { return c; }
     else { return
compareStrings(firstName,
other.firstName); }
 }
private int compareStrings(String
a, String b) {
  if (a == null) { return b == null
? 0 : 1; }
  else { return a.compareTo(b); }
  }
}
```

Java collections

`ArrayList <Object> a1 = new ArrayList <>();`	`get(int), set(int, "OO"), add("CN2"), remove(int), remove("CN1"), contains("CN1"), size()`

Java collections (cont)

`LinkedList<Object> l1 = new LinkedList<>();`	`add("ICTh"), add(int, "Bsys1"), remove(int), remove("ICTh"), contains("ICTh")`
`Set<String> s1= new TreeSet<>();` or `Set<String> s2= new HashSet<>();`	`add("Test"), remove("Test"), size()` checks if already added -> if so, return false. TreeSet = sorted, always efficient. HashSet = unsorted, usually very efficient
`Map<Integer, Object> m1= new HashMap<>();` or `Map<Integer, Object> m2= new TreeMap<>();`	`get(key), put(key, "Test"), remove(key), containsKey(key), size()`, TreeMap = sorted by key, always efficient. HashMap = unsorted, usually very efficient

```
Iterator<String> it=
m1.iterator(); while(it.hasNext())
{..}
```

Java inheritance

`Vehicle v1 = new Car();`	**Vehicle = static type, Car = dynamic type**
`Object o = new Vehicle(); Vehicle v = (Vehicle)o;`	Vaild -> Down-Cast: Static type can be down casted

By **Philip Schmid** (Higarigh)
cheatography.com/higarigh/

Published 25th January, 2015.
Last updated 15th January, 2015.

Cheatography

OOv1 Cheat Sheet

by Philip Schmid (Higarigh) via cheatography.com/20304/cs/3111/

Java inheritance (cont)

`Vehicle v = new Vehicle; Car c = (Car)v;`	Error -> Dynamic type can't be down casted
`if(v instanceof Car) { Car c = (Car)v;}`	Test if dynamic type matches static type
`super.variable`	Hiding: variable from super class
`((SuperSuperClass) this).variable`	Hiding: variable from "super.super" class

Dynamic dispatch: Methods: from dynamic typ and variables from static type.

Java Lambdas / Stream API

`Collections.sort(people, (p1, p2) -> p1.getAge() - p2.getAge());`	sort ascending by age
`Collections.sort(people, (p1, p2) -> p1.getLastName().compareTo(p2.getLastName()));`	sort by lastname
`people.stream().filter(p -> p.getAge() >= 18) .map(p -> p.getLastName()) .sorted() .forEach(System.out::println);`	stream API example

Java Lambdas / Stream API (cont)

`people.stream() .filter(p -> p.getLastName().contains(pattern)) .forEach(System.out::println);`	pattern has to be final!
`Random random= newRandom(4711); Stream.generate(random::nextInt) .forEach(System.out::println);`	generate random stream
`List<Person> list= peopleStream.collect(Collectors.toList());`	stream to collection (List/Set/Map)
`Person[] array= peopleStream.toArray(length-> newPerson[length]);`	stream to array

Possible Stream API operations: `filter(Predicate)`, `map(Function)`, `mapToInt(Function)`, `mapToDouble(Function)`, `sorted()`, `distinct()`, `limit(long n)`, `skip(long n)`, `count()`, `min()`, `max()`, `average()`, `sum()`

Comparator & Methodreference

Inferface used to compare 2 Objects(before you used lamdas).

Contains the method `public int compare(T o1, T o2)` which you need to override. Returns positiv number if o1 is bigger then o2 and negative if oppisite 0 means that they are equal

Instead of a comparator use methodrefernce class:methodName eg `PersonComp::compareName`

Nested class

Use this if a class is only used in another class

No seperate classfile

The inner class can use all members of the outer class (this include private members)

Instantiation from outside eg `Polygon.Pointpoint= myPolygon.newPoint();`

Can be declared in a method -> All variables from outside are getting final eg `Car getSuperCar() { class SuperCar extends Car { @Override public int getMaxSpeed() {return 300; } } return newSuperCar();}`

Java own exception class

```
public class MyException extends Exception {
 private static final long serialVersionUID = 1L;
 public MyException(String msg) {
   super(msg);
 }
}
```

Java package import conflict order

1. own class (inc. nested class)
2. single type imports -> `import p2.A;`
3. type in own package -> `package p1; class X`
4. Import on demand -> `import p2.*;`

By **Philip Schmid** (Higarigh)
cheatography.com/higarigh/

Published 25th January, 2015.
Last updated 15th January, 2015.

Cheatography

OOv1 Cheat Sheet

by Philip Schmid (Higarigh) via cheatography.com/20304/cs/3111/

Java regex

`1*`	0 to *
`1+`	1 to *
`1{2,5}`	min 2 max 5 (11..11111)
`1{2,}`	min 2 max * (11, 111, etc.)
`1{3}`	exactly 111
`-?1`	-1 or 1, "-" is optional
`Mo|Di`	Or
`[a-z]`	any letter from a to z
`[a-zA-Z]`	any letter from a to z or A to Z
`\s`	whitespace
`.`	anything except newline
`[^abc]`	anything except a, b or c
`$`	end of string
`\d`	any digit
`\D`	not digit
`(?<Group 1>REGEX)`	name capture group -> `String Part1 = matcher.group("Group1");`

Example: Check daytime: `([0-1]?[0-9]|2[0-3]):[0-5][0-9]`

Java regex code example

```
String input = scanner.nextLine();
Pattern pattern =
Pattern.compile("([0-2]?[0-
9]):([0-5][0-9])");
Matcher matcher =
pattern.matcher(input);
if (matcher.matches()) {
  String hoursPart =
matcher.group(1);
  String minutesPart =
matcher.group(2);
  System.out.println(..);
}
```

Java JUnit

`assertEquals(expected, actual)`	actual «equals» expected
`assertSame(expected,actual)`	actual== expected (only reference comparation)
`assertNotSame(expected, actual)`	expected != actual (only reference comparation)
`assertTrue(condition)`	condition
`assertFalse(condition)`	!condition
`assertNull(value)`	value== null
`assertNotNull(value)`	value!= null
`fail()`	everytime false
`@Test(timeout=5000)`	set test timeout
`@Test(expected=IllegalArgumentException.class)`	expect exception, if exception is thrown, test passes
`@Before public void setUp() { … }`	run this before each test
`@After public void tearDown() { … }`	run this after each test

Java JUnit examples

```
@Test
publicvoidtestPrime_2() {
  assertTrue("2 isprime",
utils.isPrime(2));
}
```

Java generics

Example: `class Node<T extends Number & Serializable>{ … }` `Node<Integer> n1; // OK` `Node<Number> n1; // OK` `Node<String> n2; // ERROR` You can add different Interfaces with & to ensure other functionality like serializable

Wildcard type: `Node<?> undefinedNode; undefinedNode = new Node<Integer>(4); undefinedNode= new Node<String>("Hi!");` No read (`.getValue()`) and write (`.setValue(X)`) is allowed

static variables with generics NOT allowed eg static T maxSpeed;

Generic Method: `public <T> T majority(T x, Ty, T z){ if(x.equals(y)) { returnx; } if(x.equals(z)) { returnx; } if(y.equals(z)) { return y; } return null; }` Call: `Double d = test.<Double>majority(1.0, 3.141, 1.0);` but also: `int i = majority(1,1,3);` called type inference(also possible to mix types of argument)

Rawtype: like you would insert Object -> you need to down cast the elements. e.g. `Node n; //without class n = new Node("Hi"); String s = (String)n.getValue();`

Serializable

Is a marker interface (is empty, just says that this class supports it)

Use it to say the developer that he can serialize objects of this class, which means he can write then in a bitecode and export them. Always serialize all the objects contained in the mainobject

Use serialVersionUID to identify your class (normally generated number) `private static final long serialVersionUID= -6583929648459736324L;`

By **Philip Schmid** (Higarigh)
cheatography.com/higarigh/

Published 25th January, 2015.
Last updated 15th January, 2015.

Serializable (cont)

Example:
```
OutputStream fos= new FileOutputStream("serial.bin") try (ObjectOutputStream stream = new ObjectOutputStream(fos)) { stream.writeObject(person); }
```

Example:
```
InputStream fis= new FileInputStream("serial.bin") try (ObjectInputStream stream = new ObjectInputStream(fis)) { Person p = (Person)stream.readObject(); … }
```

Java clone() method

```
public Department clone() throws Exception {
 Department d = new Department();
 d.name = name;
 d.people = people;
 d.subDepartments = new ArrayList<>();
 for (Department subD : subDepartments) {
  d.subDepartments.add(subD.clone());
}
 return d;
}
```

By **Philip Schmid** (Higarigh)
cheatography.com/higarigh/

Published 25th January, 2015.
Last updated 15th January, 2015.