Cheatography

Java for Testers Cheat Sheet

by matiwan via cheatography.com/33684/cs/15111/

Shortcuts IntelliJ

Show Parameters	cmd + p
Show JavaDoc	ctrl + j
Show shortcuts	cmd + j
Show popup definition	alt + space
Toggle all methods body	cmd + shift + -
Expand all methods body	cmd + shift + '+'
Todo list	Todo comment; cmd + /

JUnit

Import Junit	import org.junit.Test;

Dictionary

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called **unboxing**.

Class- fields, methods change object properties

Instantiate Class

A **static method** operates at the class level, rather than the instance or object level. Which means that we don't have to instantiate the class into a variable in order to call a static method.

Superclass

Access control

Constructor

```
access ClassName(arguments){
}
```

Invoking another constructor

```
access ClassName(arguments){
this(arguments);
}
```

Collection interface

Set	a collection that does not allow duplicates
List	a collection you can access and add elements at specific index positions
Map	a "key, value" pair where you store an object in the collection, but can access it with a unique key

Access

```
public static final String CONSTANT = "a constant string";
public static String aClassField = "a class field";
protected static String proField = "a class field";
public String pubField = "a public field";
private String privField = "a private field";
private String name;
private/public/protected/ package-private(default)
```

Type of asserts

assertEquals

Switch statement

A switch works with the byte, short, char, and int primitive data types. It also works with enumerated types (discussed in Enum Types), the String class, and a few special classes that wrap certain primitive types: Character, Byte, Short, and Integer (discussed in Numbers and Strings).
The String in the switch expression is compared with the expressions associated with each case label as if the String.equals method were being used.

Import

Static usage	import com.javafortesters.domainobject.TestAppEnv; TestEnv.method();
Static import	import static org.junit.Assert.assertEquals; ~~Assert.~~ assertEquals('a', 'b', 'c');

Inheritance

Private methods and fields are not accessible through inheritance, only the super class's **protected** and **public** fields and methods are accessible through inheritance.

JavaDoc

```
@Test
public void aJavaDocComment(){
assertTrue(addTwoNumbers(4,3)==7);
}
    /**
     * Add two integers and return an int.
     *
     * There is a risk of overflow since two big
     * integers would max out the return int.
     *
     * @param a is the first number to add
     * @param b is the second number to add
     * @return a+b as an int
     */
public int addTwoNumbers(int a, int b){ return a+b;
}
//click at the name of function and press ctrl+j
```

Java for Testers Cheat Sheet

Good practices

When you encounter:
• any Java library that you don’t know how to use
• parts of Java that you are unsure of
• code on your team that you didn’t write and don’t understand
Then you can:
• read the documentation - ctrl + q (ctrl + j on Mac) or on-line web docs
• read the source - ctrl and click on the method, to see the source
• write some @Test annotated methods, with assertions, to help you explore the functionality of the library
When writing the @Test methods you need to keep the following in mind:
• write just enough code to trigger the functionality
• ensure you write assertion statements that cover the functionality well and are readable
• experiment with ‘odd’ circumstances

Design Inheritance

Inheritance
```
class Fruit {
//...
}
class Apple extends Fruit {
//...
}
```

Design Composition

using instance variables that are references to other objects
```
class Fruit {
//...
```

Design Composition (cont)

```
}
class Apple {
private Fruit fruit = new Fruit();
//...
}
```
In a composition relationship, the front-end class holds a reference in one of its instance variables to a back-end class.

https://www.artima.com/designtechniques/compoi nh3.html