## List

### Clone

Not support by default, use outside libs (reflection).

<https://stackoverflow.com/questions/2156120/java-recommended-solution-for-deep-cloning-copying-an-instance>

For deep cloning (clones the entire object hierarchy):

commons-lang SerializationUtils - using serialization - if all classes are in your control and you can force implementing Serializable.

Java Deep Cloning Library - using reflection - in cases when the classes or the objects you want to clone are out of your control (a 3rd party library) and you can't make them implement Serializable, or in cases you don't want to implement Serializable.

For shallow cloning (clones only the first level properties):

commons-beanutils BeanUtils - in most cases.

Spring BeanUtils - if you are already using spring and hence have this utility on the classpath.

I deliberately omitted the "do-it-yourself" option - the API's above provide a good control over what to and what not to clone (for example using transient, or String[] ignoreProperties), so reinventing the wheel isn't preferred.

<https://github.com/kostaskougios/cloning/>

Here is an example of its usage:

Cloner cloner=new Cloner();

MyClass clone=cloner.deepClone(o);

// clone is a deep-clone of o

#For first level implements Clonable and override clone()

@Override

protected Person clone() {

Person clone = null;

try {

clone = (Person) super.clone();

} catch (CloneNotSupportedException e) {

e.printStackTrace();

}

return clone;

}

## Properties

<https://docs.oracle.com/javase/tutorial/essential/environment/properties.html>

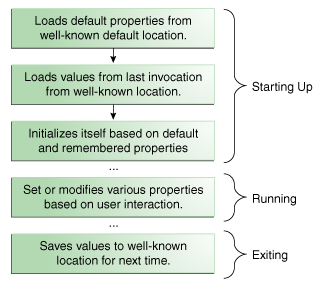
How to setup, config and use properties?

### Properties object

*key/value pairs*, both [String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html) values.

Manage by [java.util.Properties](https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html) extends [java.util.Hashtable](https://docs.oracle.com/javase/8/docs/api/java/util/Hashtable.html). Provides methods:

* loading key/value pairs into a Properties object from a **stream**,
* retrieving a value from its key,
* listing the keys and their values,
* enumerating over the keys, and
* saving the properties to a stream.



* Setting Up

// create and load default properties

Properties defaultProps = new Properties();

FileInputStream in = new FileInputStream("defaultProperties");

defaultProps.load(in);

in.close();

// create application properties with default

Properties applicationProps = new Properties(defaultProps);

// now load properties

// from last invocation

in = new FileInputStream("appProperties");

applicationProps.load(in);

in.close();

* Saving Properties

FileOutputStream out = new FileOutputStream("appProperties");

applicationProps.store(out, "---No Comment---");

out.close();

**Note:** Some of the methods described above are defined in Hashtable, and thus accept key and value argument types other than String. **Always use Strings for keys and values**, even if the method allows other types. Also do not invoke Hashtable.set or Hastable.setAll on Properties objects; always use **Properties.setProperty**.

### System Properties

<https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html>

|  |  |
| --- | --- |
| **Key** | **Meaning** |
| "file.separator" | Character that separates components of a file path. This is "/" on UNIX and "\" on Windows. |
| "java.class.path" | Path used to find directories and JAR archives containing class files. Elements of the class path are separated by a platform-specific character specified in the path.separator property. |
| "java.home" | Installation directory for Java Runtime Environment (JRE) |
| "java.vendor" | JRE vendor name |
| "java.vendor.url" | JRE vendor URL |
| "java.version" | JRE version number |
| "line.separator" | Sequence used by operating system to separate lines in text files |
| "os.arch" | Operating system architecture |
| "os.name" | Operating system name |
| "os.version" | Operating system version |
| "path.separator" | Path separator character used in java.class.path |
| "user.dir" | User working directory |
| "user.home" | User home directory |
| "user.name" | User account name |

The System class has two methods used to read system properties: getProperty and getProperties.

#### Reading

System.getProperty("path.separator");

System.getProperty("subliminal.message", "Buy StayPuft Marshmallows!");

#### Writing

Add properties from “myProperties.txt” to System properties.

import java.io.FileInputStream;

import java.util.Properties;

public class PropertiesTest {

public static void main(String[] args)

throws Exception {

// set up new properties object

// from file "myProperties.txt"

FileInputStream propFile =

new FileInputStream( "myProperties.txt");

Properties p =

new Properties(System.getProperties());

p.load(propFile);

// set the system properties

System.setProperties(p);

// display new properties

System.getProperties().list(System.out);

}

}

**Note**

* The setProperties method, changes are not persistent. Will not affect future invocations of the Java interpreter for this or any other application.
* If changes are to be persistent, then write to some file before exiting and read again upon startup.

## Properties in actions

# Regex

<https://stackoverflow.com/questions/15409296/what-is-the-use-of-pattern-quote-method/15409355#15409355>

TL;DR

someString.matches(Pattern.quote(someLiteral));

someString.matches("\\Q" + someLiteral + "\\E"));

Returns a literal pattern String for the specified String.

This method produces a String that can be used to create a Pattern that would match the string s as if it were a literal pattern.

Metacharacters or escape sequences in the input sequence will be given no special meaning.

System.out.println("foo".matches(".\*")); // true

System.out.println("foo".matches(Pattern.quote(".\*"))); // false

System.out.println(".\*".matches(Pattern.quote(".\*"))); // true