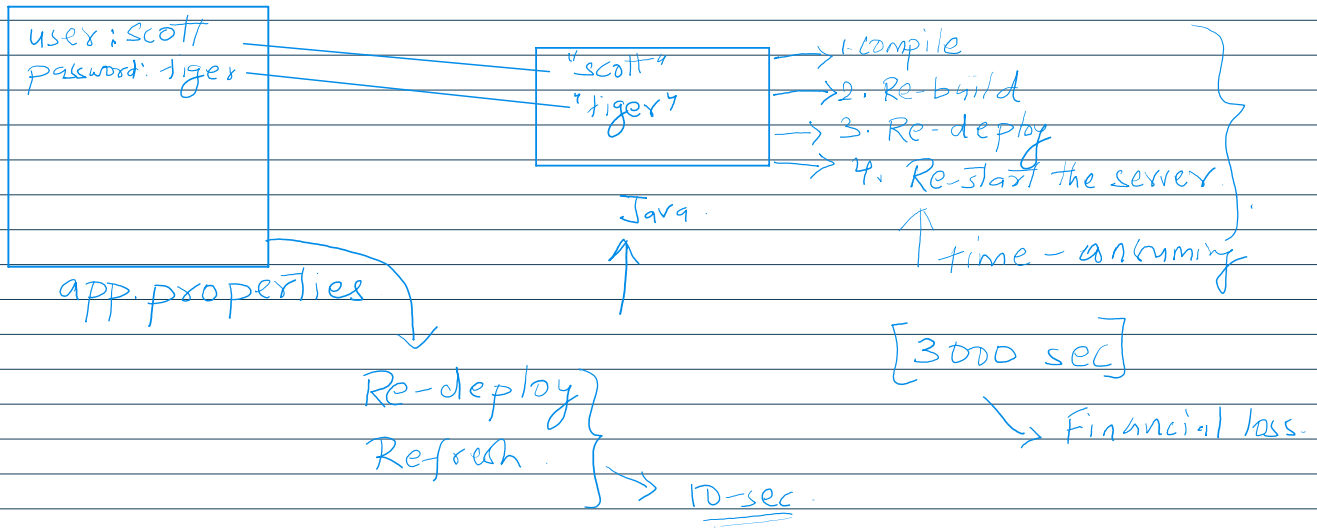
Properties

05 June 2025 20:13



In our program, if anything, which changes frequently (like user name, password, mail ids, mobile number, etc.) are not recommended to hard code (Keeping the values directly) in the Java program. Because if there is any change to reflect that change, recompilation, rebuild, redeploy applications are required, even sometimes servers restart also required, which creates business impact to the client.

We can overcome this problem using properties file. Such type of variable thing we have to configure in the properties file that we have to read into Java program. And we can use those properties.
The main advantage of this approach is if there is a change in properties file to reflect that change, just redeployment is enough, which would not create any business impact to the client.

We can use Java **Properties** object to hold properties which are coming from properties file.

```
Properties p = new Properties();
```

In normal map (like HashMap, Hashtable, TreeMap) key-value can be any type. But in case of Properties, key-value should be String type.

Methods:

```
String setProperty(String pname, String pvalue);
```

Old value — If the specified property already available, then old value will be replaced with new values and returns the old value. This function is used to set a new property.

```
String getProperty(String pname);
```

To get the value associated with the specified property. If the specified property is not available, then this method returns null.

```
Enumeration propertyNames();
```

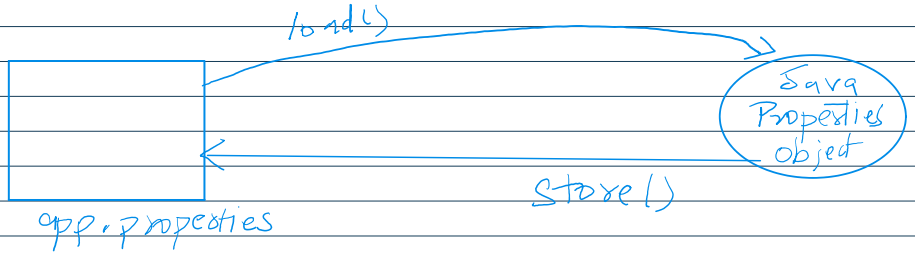
Returns all property name present in properties object.

```
void load(InputStream is);
```

To load properties from properties filed into Java properties object.

```
void store(OutputStream os);
```

To store proper ties from Java. Properties object into properties file.



Official Documentation:

public class Properties
extends Hashtable<Object,Object>

The Properties class represents a persistent set of properties. The Properties can be saved to a stream or loaded from a stream. Each key and its corresponding value in the property list is a string.

A property list can contain another property list as its "defaults"; this second property list is searched if the property key is not found in the original property list.

Because Properties inherits from Hashtable, the put and putAll methods can be applied to a Properties object. Their use is strongly discouraged as they allow the caller to insert entries whose keys or values are not Strings. The setProperty method should be used instead. If the store or save method is called on a "compromised" Properties object that contains a non-String key or value, the call will fail. Similarly, the call to the propertyNames or list method will fail if it is called on a "compromised" Properties object that contains a non-String key.

The iterators returned by the iterator method of this class's "collection views" (that is, entrySet(), keySet(), and values()) may not fail-fast (unlike the Hashtable implementation). These iterators are guaranteed to traverse elements as they existed upon construction exactly once, and may (but are not guaranteed to) reflect any modifications subsequent to construction.

The load(Reader) / store(Writer, String) methods load and store properties from and to a character based stream in a simple line-oriented format specified below. The load(InputStream) / store(OutputStream, String) methods work the same way as the load(Reader)/store(Writer, String) pair, except the input/output stream is encoded in ISO 8859-1 character encoding. Characters that cannot be directly represented in this encoding can be written using Unicode escapes as defined in section 3.3 of The Java Language Specification; only a single 'u' character is allowed in an escape sequence.

The loadFromXML(InputStream) and storeToXML(OutputStream, String, String) methods load and store properties in a simple XML format. By default the UTF-8 character encoding is used, however a specific encoding may be specified if required. Implementations are required to support UTF-8 and UTF-16 and may support other encodings. An XML properties document has the following DOCTYPE declaration:

```
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
```

Note that the system URI (http://java.sun.com/dtd/properties.dtd) is not accessed when exporting or importing properties; it merely serves as a string to uniquely identify the DTD, which is:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- DTD for properties -->
```

```
    <!ELEMENT properties ( comment?, entry* ) >

    <!ATTLIST properties version CDATA #FIXED "1.0">

    <!ELEMENT comment (#PCDATA) >

    <!ELEMENT entry (#PCDATA) >

    <!ATTLIST entry key CDATA #REQUIRED>
```

This class is thread-safe: multiple threads can share a single Properties object without the need for external synchronization.

API Note:
The Properties class does not inherit the concept of a load factor from its superclass, Hashtable.

Since:
1.0

Constructor Summary

Constructors

Constructor	Description
`Properties()`	Creates an empty property list with no default values.
`Properties(int initialCapacity)`	Creates an empty property list with no default values, and with an initial size accommodating the specified number of elements without the need to dynamically resize.
`Properties(Properties defaults)`	Creates an empty property list with the specified defaults.

All Methods | **Instance Methods** | **Concrete Methods** | **Deprecated Methods**

Modifier and Type	Method	Description
`String`	`getProperty(String key)`	Searches for the property with the specified key in this property list.
`String`	`getProperty(String key, String defaultValue)`	Searches for the property with the specified key in this property list.
`void`	`list(PrintStream out)`	Prints this property list out to the specified output stream.
`void`	`list(PrintWriter out)`	Prints this property list out to the specified output stream.
`void`	`load(InputStream inStream)`	Reads a property list (key and element pairs) from the input byte stream.
`void`	`load(Reader reader)`	Reads a property list (key and element pairs) from the input character stream in a simple line-oriented format.
`void`	`loadFromXML(InputStream in)`	Loads all of the properties represented by the XML document on the specified input stream into this properties table.
`Enumeration<?>`	`propertyNames()`	Returns an enumeration of all the keys in this property list, including distinct keys in the default property list if a key of the same name has not already been found from the main properties list.
`void`	`save(OutputStream out, String comments)`	**Deprecated.** This method does not throw an IOException if an I/O error occurs while saving the property list.
`Object`	`setProperty(String key, String value)`	Calls the `Hashtable` method `put`.

void	store(OutputStream out, String comments)	Writes this property list (key and element pairs) in this Properties table to the output stream in a format suitable for loading into a Properties table using the load(InputStream) method.
void	store(Writer writer, String comments)	Writes this property list (key and element pairs) in this Properties table to the output character stream in a format suitable for using the load(Reader) method.
void	storeToXML(OutputStream os, String comment)	Emits an XML document representing all of the properties contained in this table.
void	storeToXML(OutputStream os, String comment, String encoding)	Emits an XML document representing all of the properties contained in this table, using the specified encoding.
void	storeToXML(OutputStream os, String comment, Charset charset)	Emits an XML document representing all of the properties contained in this table, using the specified encoding.
Set<String>	stringPropertyNames()	Returns an unmodifiable set of keys from this property list where the key and its corresponding value are strings, including distinct keys in the default property list if a key of the same name has not already been found from the main properties list.

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;
public class PropertiesDemo {
    public static void main(String[] args) {
        Properties p = new Properties();
        try {
            p.load(new FileInputStream("./CollectionFramework/Map/app.properties"));
            String str = p.getProperty("user");
            System.out.println("value: " + str);
            p.setProperty("newuser2", "shivin");
            p.setProperty("newpassword2", "12345@09876abc");
            p.store(new FileOutputStream("./CollectionFramework/Map/app.properties"), "Just
updated");
        }
        catch(FileNotFoundException e){
            System.out.println("File nahi mila!!!!"+"\n"+e.getMessage());
        }
        catch (IOException e) {
            // TODO Auto-generated catch block
            System.out.println("Network Error"+"\n"+e.getMessage());
        }
    }
}
```

Output:

```
value: raghav
```

CollectionFramework > Map > app.properties

```
1  #Just updated
2  #Thu Jun 05 20:55:17 IST 2025
3  newpassword1=12345@09876
4  newpassword2=12345@09876abc
5  newuser1=shiv
6  newuser2=shivin
7  password=12345@09876
8  user=raghav
9
```