Lock l = ...;

 l.lock();

 try {

 // access the resource protected by this lock

 } finally {

 l.unlock();

 }

**tryLock**

boolean **tryLock**()

Acquires the lock only if it is free at the time of invocation.

Acquires the lock if it is available and returns immediately with the value true. If the lock is not available then this method will return immediately with the value false.

A typical usage idiom for this method would be:

 Lock lock = ...;

 if (lock.tryLock()) {

 try {

 // manipulate protected state

 } finally {

 lock.unlock();

 }

 } else {

 // perform alternative actions

 }

This usage ensures that the lock is unlocked if it was acquired, and doesn't try to unlock if the lock was not acquired.

**Returns:**

true if the lock was acquired and false otherwise.

package lock;

import java.util.concurrent.locks.ReentrantLock;

public class Livro extends ReentrantLock implements Comparable

{

 private int id;

 private int timeReading;

 private int readers;

 private int timeToRead;

 private BufferMensagem buffer;

 public Livro(int codigo, int tempoDeLeitura, int quantidadeDeLeitoresSimultaneos,
 BufferMensagem buffer)

 {

 super();

 this.id = codigo;

 this.timeReading = tempoDeLeitura;

 this.readers = quantidadeDeLeitoresSimultaneos;

 this.timeToRead = 0;

 this.buffer = buffer;

 }

 public boolean emprestar()

 {

 If ( this.tryLock() )

 {

 this.buffer.mensagem(String.format("O livro " + this.getCodigo() + " foi emprestado ao
 estudante: " + Thread.currentThread().getId() + " \n"));

 return true;

 }

 return false;

 }

 public void devolver()

 {

 this.unlock();

 this.buffer.mensagem(String.format("O livro " + id + " foi devolvido pelo estudante: " +
 Thread.currentThread().getId() + " \n" ));

 }

 public void ler()

 {

 try

 {

 Thread.sleep(timeReading);

 this.timeToRead++;

 }

 catch (Exception excecao)

 {

 System.out.println(excecao.getMessage());

 }

 }

 public int getCodigo()

 {

 return id;

 }

 public int getTempoDeLeitura()

 {

 return timeReading;

 }

 public int compareTo (Object o)

 {

 int valorDeRetorno = 0;

 if(o instanceof Livro)

 {

 Livro livro = (Livro) o;

 If (livro.getCodigo() == this.getCodigo())

 {

 valorDeRetorno = 1;

 }

 }

 return valorDeRetorno;

 }

 public int getTempoLido()

 {

 return timeToRead;

 }

}

|  |
| --- |
| **public**Object clone() |

Cloning a vector with clone() is like making a shallow copy of the vector.

A new vector is created with each object reference copied from the original vector.

Similar to calling the Vector constructor that accepts a Collection.

All elements of the two vectors will effectively point to the same set of objects.

|  |
| --- |
| **import**java.util.Collections;**import**java.util.Vector;**public class**MainClass {  **public static void**main(String args[]) {    Vector v1 = **new**Vector();    v1.add("A");    v1.add("C");    v1.add("B");    Vector v2 = (Vector) v1.clone();    Collections.sort(v2);    System.out.println(v1);    System.out.println(v2);  }} |

## java.lang Class Math

[java.lang.Object](http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Object.html)

 **java.lang.Math**

public final class **Math**

extends [Object](http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Object.html)

The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

**Method random**

public static double **random**()

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression

new java.util.Random

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else.

This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom-number generator.