```
public class Matrix {

private double[][]mat;
//AF(c)={c.m[i][j]}

//@private invariant mat!=null && mat.length >0
//@ && (\forall int i; 0<=i<mat.length; mat[i]!=null)
//@ && (\forall int r1; 0<=r1<mat.length;
//@      (\forall int r2; 0<=r2<mat.length; mat[r1].length == mat[r2].length));

//CREATORs
//@ ensures rows>0 && cols>0 &&  rows()==rows && ncols()==cols &&
//@    (\forall int i; 1<=i<=nrows();
//@           (\forall int j; 1<=j<=ncols(); el(i,j)==0));
//@ signals (MatrixException e) rows==0 || cols==0;
public Matrix(int rows, int cols) throws MatrixException  {
      mat = new double[rows][cols];
}

/*@ensures data!=null && data.length!=0 && data[0]!=null && data[0].length!=0 &&
  @ (\forall int i; 1<=i<data.length;
  @   data[i]!=null && data[i].length==data[0].length) &&
  @ nrows()==data.length && ncols()==data[0].length &&
  @ (\forall int i; 1<=i<=nrows();
  @  (\forall int j; 1<=j<=ncols();
  @   el(i,j)==data[i-1][j-1]));
  @signals (MatrixException e) data==null || data.length==0
  @ ||(\exists int i; 0<=i<data.length; data[i]==null)
  @ ||data[0].length==0
  @ ||(\exists int i; 0<=i<=data.length;
  @    (\exists int j; 0<=j<=data.length;
  @     data[i].length != data[j].length));
  @*/
public Matrix(double[][] data) throws MatrixException {
      //scorretto:  mat = data;  (esporrebbe la rappresentazione interna)

      //corretto
  if (data == null || data.length == 0)
    throw new MatrixException("empty dataset");

  for (int i = 0; i < data.length; i++) {
    if (data[i] == null || data[i].length == 0)
       throw new MatrixException("Empty column ("+i+")");

    for (int j = 0; j < data.length; j++) {
      if (data[i].length != data[j].length)
        throw new MatrixException("Different Columns length");
    }
  }

  mat = new double[data.length][data[0].length];
  for (int i = 0; i < data.length; i++) {
    for (int j = 0; j < data[i].length; j++) {
      mat[i][j] = data[i][j];
    }
  }
}

//OBSERVERs
/*@ensures 0<r<=nrows() && 0<c<=ncols() && (*\result == elemento in posizione
r,c *);
  @signals (MatrixException e) r<=0 || r>nrows() || c<=0 || c>ncols(); @*/
```

```
public /*@pure@*/ double el(int r, int c)  throws MatrixException
{
  if (r<=0 || r > mat.length || c<=0 || c > mat[0].length)
    throw new MatrixException("Index out ");

  return mat[r-1][c-1];  //in Java gli array partono da 0; le matrici partono da
(1,1)
}

/*@ensures0<c<=ncols() &&  \result.length==nrows() &&
   @              (\forall int i; 0<=i<\result.length;  \result[i]==el(i+1,c));
   @ signals (MatrixException e) c<=0 || c>ncols(); @*/
public /*@pure@*/ double[] col(int c)  throws MatrixException
{
  if (c<=0 || c>mat[0].length) throw new MatrixException("...");

  double[] res = new double[mat.length];
  for (int i=0; i < mat.length; i++)  res[i] = mat[i][c-1];
  return res;
}

/*@ensures 0<r<=nrows() &&   \result.length==ncols() &&
  @(\forall int i; 0<=i<\result.length;  \result[i]==el(r,i+1));
  @ signals (MatrixException e) r<=0 || r>nrows(); @*/
public /*@pure@*/ double[] row(int r) throws MatrixException
{
  //scorretto:  return mat[r-1];  (esporrebbe la rappresentazione interna)

  // corretto:
  if (r<=0 || r>mat.length) throw new MatrixException("...");

  double[] res = new double[mat[0].length];
  for (int i=0; i < mat[0].length; i++) res[i] = mat[r-1][i];
  return res;
}

/*@ensures (forall int i; 0<=i<\result.length;
  @   (\forall int j; 0<=j<\result[i].length; \result[i][j] == el(i+1,j+1)));
  @*/
public /*@pure@*/ double[][] toArray()
{
  double[][] res = new double[mat.length][mat[0].length];

  for (int i=0; i<mat.length; i++)
    for (int j=0; j<mat[0].length; j++)
      res[i][j] = mat[i][j];

  return res;
}

/*@ensures (*\result==numero righe di this *);@*/
public /*@pure@*/ int nrows()
{ return mat.length; }

/*@ensures (*\result==num. colonne di this *);@*/
public /*@pure@*/ int ncols()
{ return mat[0].length; }

/*@ensures (*\result==rango di this *);@*/
public /*@pure@*/ int rank();

//PRODUCERs
```

```
/*@requires m!=null && nrows()==m.nrows() && ncols==m.ncols();
   ensures nrows()==\result.nrows() &&  ncols()==\result.ncols() &&
    (\forall int i; 1<=i<nrows(); (\forall int j; 1<=j<=ncols();
\result.el(i,j)== el(i,j)+m.el(i,j))); @*/
public Matrix sub(Matrix m);
public Matrix sum(Matrix m)
{ /* qui, controllo sulle dimensioni e lancio eccezioni,  con specifiche
totali*/

  /* soluzione alternativa: matrice vuota di dimensioni opportune, riempita con
il mutator setEl */
  double[][] res = new double[mat.length][mat[0].length];

  for (int i=0; i<mat.length; i++)
    for (int j=0; j<mat.length; j++)
      res[i][j] = mat[i][j] + m.mat[i][j];

  return new Matrix(res);
}

//@ requires m !=null && this.ncols()==m.nrows();
//@ ensures \result != null &&
//@         \result.nrows() == this.nrows() && \result.ncols() == m.ncols() &&
//@         (\forall int i; 1 <= i < this.nrows();
//@           (\forall int j; 1 <= j < m.ncols();
//@             \result.el(i,j) ==
//@               (\sum int k; 1 <= k < this.ncols(); this.el(i,k) * m.el(k,j) )
//@         ) );
public Matrix mult(Matrix m)
{ double[][] res = new double[mat.length][m.mat[0].length];
  for (int i=0; i<mat.length; i++)
   for (int j=0; j<m.mat[0].length; j++) {
    res[i][j] = 0;
    for (int k=0; k<mat[0].length; k++)
     res[i][j] += mat[i][k]*m.mat[k][j];
   }
  return new Matrix(res);
}

//@ ensures ncols()==\result.nrows() &&  nrows()==\result.ncols() &&
//@  (\forall int i; 1<=i<nrows();  (\forall int j; 1<=j<=ncols();
\result.el(j,i)== el(i,j)));
public Matrix transpose()
{
 double[][] res = new double[mat[0].length][mat.length];

 for (int i=0; i<mat.length; i++)
  for (int j=0; j<mat.length; j++)
   res[j][i] = mat[i][j];

 return new Matrix(res);
}

//MUTATORs
//@ensures 0<r && r<=nrows() && 0<c && c<=ncols()   && el(r,c) == val;
//@signals (MatrixException e) r<=0 || r>nrows() || c<=0 || c>ncols();
public void setEl(double val, int r, int c) throws MatrixException {
  if (r<=0 || c<=0 || r>mat.length || c>mat[0].length)
    throw new MatrixException("Out of bounds");

  mat[r-1][c-1] = val;
}
```

```
// UTILITY

// equals: va bene quello di Object perchè è mutabile.
// Facciamo un metodo di utilità (sameElements) che dica se due matrici
contengono gli stessi numeri

/*@ensures \result == true <==>
  @ nrows() == m.nrows() && ncols() == m.ncols() &&
  @ (\forall int i; 1<=i<=nrows();
  @    (\forall int j; 1<=j<=ncols(); el(i,j) == m.el(i,j)));
  @*/
public /*@ pure @*/ boolean sameElements(Matrix m) {
 if (mat.length  != m.mat.length || mat[0].length  != m.mat[0].length)
   return false;

 for (int i = 0; i < mat.length; i++)
  for (int j = 0; j < mat[0].length; j++)
   if (mat[i][j] != m. mat[i][j]) return false;

 return true;
}

public Object clone() {
 Matrix res = null;
 try {
  res = new Matrix(mat);  //ok: il costruttore fa copia di mat => no
condivisione rep.
 } catch(MatrixException me) {
  //qui mai, dato che è rispettato il rep invariant.
 }
 return res;
}

public String toString() {
  StringBuffer sb = new StringBuffer();

  for (int i = 0; i < mat.length; i++) {
    for (int j = 0; j < mat[0].length; j++) {
      sb.append(mat[i][j]+" ");
    }
    sb.append("\n");
  }
  return sb.toString();
}


} // end class
```