Exercise 11
– Refactoring (Solution) –

The refactoring process can be described as the following sequence of steps:

- Essentially we refactor the loop body of the method *addAll*. It can be schematically represented in the following way:

```
if(M1 instanceof FullMatrix)
  if(M2 instanceof FullMatrix)
    AddOp1(M1, M2);
  else
    AddOp2(M1, M2);
else
  if(M2 instanceof FullMatrix)
    AddOp3(M1, M2);
  else
    AddOp4(M1, M2);
```

  Here M1 and M2 are elements of arrays of matrices which are currently processed by the loop body. $AddOp_i$, where $i \in [1..4]$, are different kind of addition operations.

- To get rid of the outer *if* statement. We add method $add(MatrixM)$ to classes **Matrix**, **DiagonalMatrix**, **FullMatrix**. They can be schematically represented in the following way:

```
class Matrix{
  public abstract Matrix add(Matrix m);
}

class FullMatrix{
  public Matrix add(Matrix m){
    if(m instanceof FullMatrix)
      AddOp1(this, m);
    else
      AddOp2(this, m);
  }
}

class DiagonalMatrix{
  public Matrix add(Matrix m){
    if(m instanceof FullMatrix)
      AddOp3(this, m);
    else
```

```
      AddOp4(this, m);
  }
}
```

After addition of these methods we can replace the loop body by M1.*add*(M2).

- Now we would like to get rid of the remaining *if* statement in both *add* methods of **DiagonalMatrix** and **FullMatrix**. To do it we add methods *addDiagonal* and *addFull* to classes **Matrix**, **DiagonalMatrix**, and **FullMatrix**. We use these methods to determine the dynamic type of m. To do it we use equivalence of $A + B$ and $B + A$ to change order of operands of $AddOp_i$, where $i \in [1..4]$. The source code after these transformations can be scratched as follows:

```
class Matrix{
  public abstract Matrix add(Matrix m);
  protected abstract Matrix addDiagonal(DiagonalMatrix m);
  protected abstract FullMatrix addFull(FullMatrix m);
}

class FullMatrix{
  public Matrix add(Matrix m){
    m.addFull(this);
  }

  protected Matrix addDiagonal(DiagonalMatrix m){
    AddOp3(this, m);
  }

  protected FullMatrix addFull(FullMatrix m){
    AddOp1(this, m);
  }
}

class DiagonalMatrix{
  public Matrix add(Matrix m){
    m.addDiagonal(this);
  }

  protected Matrix addDiagonal(DiagonalMatrix m){
    AddOp4(this, m);
  }

  protected FullMatrix addFull(FullMatrix m){
    AddOp2(this, m);
```

```
    }
}
```

- We can see that $AddOP_2$ and $AddOP_3$ are semantically equivalent. We would like to eliminate this redundancy. To do it we change body of method *addFull* of class **DiagonalMatrix** to $AddOp2(m, this)$ (we can do it because $A + B$ is equivalent to $B + A$). And finally we replace it by $m.addDiagonal(this)$. Now instead of the code duplication we have the method call.

The refactored version of the source code is the following:

```
abstract class Matrix {
  public abstract Matrix add(Matrix m);
  protected abstract Matrix addDiagonal(DiagonalMatrix m);
  protected abstract FullMatrix addFull(FullMatrix m);
}

class FullMatrix extends Matrix{
  public Matrix add(Matrix m){
    assert(getSize() == m.getSize());
    return m.addFull(this);
  }

  protected FullMatrix addDiagonal(DiagonalMatrix m){
    FullMatrix result = new FullMatrix(this);
      for(int i=0; i<size;i++)
        result.elements[i][i] += m.get(i,i);
      return result;
  }

  protected FullMatrix addFull(FullMatrix m) {
    FullMatrix result = new FullMatrix(this);
      for(int i=0; i<size;i++)
        for(int j=0; j<size;j++)
          result.elements[i][j] += m.elements[i][j];
    return result;
  }
}

class DiagonalMatrix extends Matrix{
  public Matrix add(Matrix m){
    assert(getSize() == m.getSize());
    return m.addDiagonal(this);
```

```
  }

  protected DiagonalMatrix addDiagonal(DiagonalMatrix m){
    DiagonalMatrix result = new DiagonalMatrix(this);
    for(int i=0; i<size;i++)
      result.elements[i] += m.elements[i];
    return result;
  }

  protected FullMatrix addFull(FullMatrix m){
    return m.addDiagonal(this);
  }
}

public class ArrayOfMatrix {
  public ArrayOfMatrix addAll(ArrayOfMatrix ar){
    assert(size == ar.size);
    ArrayOfMatrix result = new ArrayOfMatrix(size);
    for(int ind=0; ind<size; ind++)
      result.elements[ind] = elements[ind].add(ar.elements[ind]);
    return result;
  }
}
```