

CMPT225, Spring 2025
Homework Assignment 1
Due date: Friday, September 26, 2025, 23:59

You need to implement the following classes:

Part 1:

- *integeriterators.ArrayIterator.java*
- *integeriterators.RangeIterator.java*
- *integeriterators.PrimeNumbersIterator.java*

Part 2:

- *rubikscube.RubiksCube.java*

Note that the files for Part 1 must be in under the package (folder) **integeriterators/**, and the files for Part 2 must be under the package (folder) **rubikscube/**.

You may add more classes to your solution if necessary.

Submit all your files in **assignment1.zip** to CourSys. Make sure your zip file can be unzipped using the command “*unzip assignment1.zip*” in CSIL. All your solutions in the zip file must be under the **src** folder, same as in the provided *cmpt225-hw1-fall25.zip*.

Compilation: Your code MUST compile in CSIL using javac.

Make sure that your code compiles without warnings/errors.

If the code does not compile in CSIL the grade on that part will be 0 (zero).

Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, please specify the references in comments. Asking others for solutions (online or in person) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Do not add main() to your solutions. The main() method will be in the test files.

Examples of such tests are provided in *TestIterators.java* and *TestRubiksCube.java*.

Warnings: Warnings during compilation will reduce points. More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [50 points] - IntegerIterator **extends** Iterator<Integer>

IntegerIterator represents a sequence (finite or infinite) of integers.
The interface has the following public methods:

public boolean hasNext() : Returns true if the sequence has more elements, and returns false otherwise. If the sequence is infinite, hasNext() always returns true.
It is inherited from Iterator<Integer>

public Integer next() : Returns the next element in the sequence.
It is inherited from Iterator<Integer>

public void reverse() : Reverses the direction of the iterator.

The interface is provided with the assignment under the package integeriterators.

Your goal is to write the following 3 classes implementing this interface:

1) **ArrayIterator [15 points]**

The class has the following constructor.

public ArrayIterator(int[] ar)

The constructor gets an array of ints. For this constructor the iterator goes through the elements of the array in the given order starting from **ar[0]**.

*Pay attention to the functionality of the **reverse()** method. For example, on the array [10,20,30], the following results are expected.

```
next() ; -- returns 10
next() ; -- returns 20
next() ; -- returns 30
hasNext() ; -- returns False
reverse() ;
hasNext() ; -- returns True
next() ; -- returns 20
reverse() ;
next() ; -- returns 30
hasNext() ; -- returns False
reverse() ;
hasNext() ; -- returns True
next() ; -- returns 20
```

2) RangeIterator [15 points]

The class has two constructors:

1) `public RangeIterator(int n)`

The iterator iterates through the elements 0,1...n-1.

If $n \leq 0$, throws `IllegalArgumentException`.

2) `public RangeIterator(int a, int n)`

Defines the sequence $a, a+1, a+2, \dots, a+n-1$.

If $n \leq 0$, throws `IllegalArgumentException`.

3) PrimeNumbersIterator [20 points]

The class implements the infinite sequence of prime numbers. It has two constructors:

1) `public PrimeNumbersIterator()`

Defines the sequence of all prime numbers in the increasing order: 2,3,5,7,11,13,17,19... (0 and 1 are not prime numbers). For example, the following results are expected:

`next()` ; -- returns 2

`next()` ; -- returns 3

`next()` ; -- returns 5

`hasNext()` ; -- returns `True`

`reverse()` ;

`next()` ; -- returns 3

`next()` ; -- returns 2

`hasNext()` ; -- returns `False`

`reverse()` ;

`next()` ; -- returns 3

2) `public PrimeNumbersIterator(int n)`

Creates a prime number iterator that starts from the n'th prime number.

For example, when creating the iterator using `PrimeNumbersIterator(3)`, the following results are expected:

`next()` ; -- returns 5, because 5 is the fourth prime

`next()` ; -- returns 7

`reverse()` ;

`next()` ; -- returns 5

`next()` ; -- returns 3

`next()` ; -- returns 2

`hasNext()` ; -- returns `False`

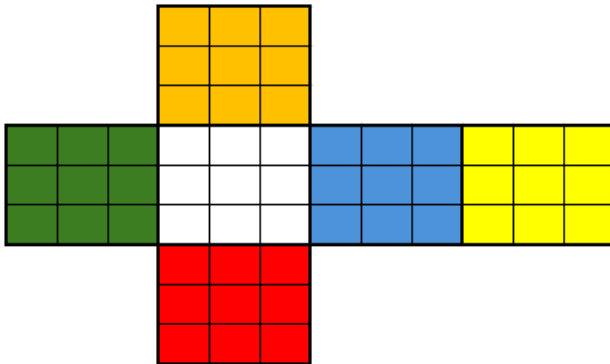
`reverse()` ;

`next()` ; -- returns 3

Part 2 [50 points] - Rubik's Cube

RubiksCube is a class representing the puzzle Rubik's Cube. In this exercise you will read the cube from a file, make a sequence of moves, and then store the result in a different file.

Below is the representation of the solved rubik's cube unfolded to a 2D plane.



This state is represented in the file `cube_init.txt`.

The tiles of the cube have 6 colors: **White, Yellow, Blue, Green, Orange, Red**. The colors are represented in the file using the letters Y,W,B,G,R,O.

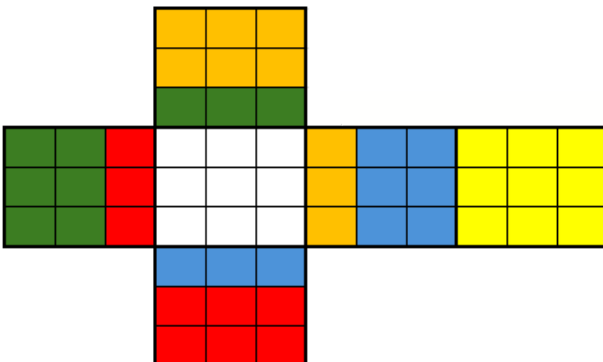
A state of the cube in a file is represented using 9 rows.

- The first three rows start with three empty spaces followed by three colors
- The next three rows have 3x4=12 colors
- The last three rows start with three empty spaces followed by three colors.

In this representation

- White color is the **F**ront.
- Yellow is the **B**ack.
- Blue is the **R**ight.
- Green is the **L**eft.
- Orange is the **U**p.
- Red is the **D**own.

A move corresponds to one of the letters: **F,B,R,L,U,D**, which represents the rotation of one of the sides *clockwise*. For example, if we start from the state above, the move "**F**" corresponds to rotating the front side clockwise, and the result will be:



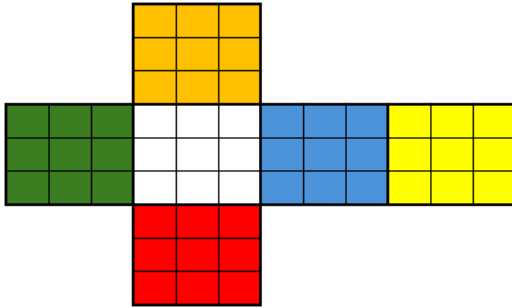
This state is represented in the file cube_f.txt

Write the class *RubiksCube* with the following constructor and public methods:

The class has two constructors:

```
public RubiksCube()
```

The constructor creates the cube with the default state



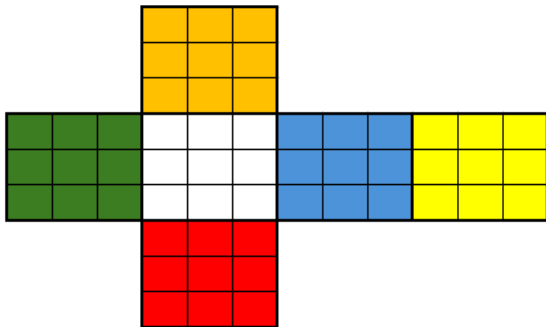
```
public RubiksCube(String fileName) throws IOException,  
IncorrectFormatException
```

The constructor gets a name of a file as an argument, reads the file and initializes the cube. It throws an exception if the file does not exist or if the format of the file is not formatted properly. The class `IncorrectFormatException` is provided with the assignment.

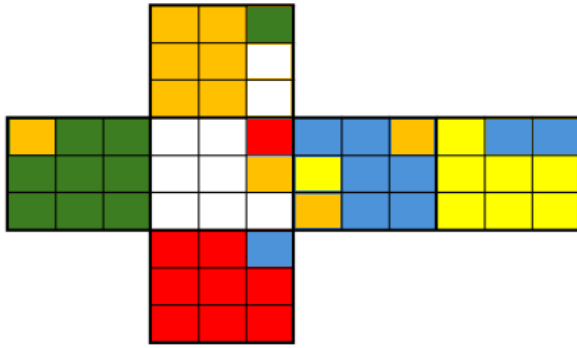
1)

```
public void applyMoves(String moves)
```

The method gets a sequence of moves, and changes the state of the cube according to the sequence. The sequence of moves is given as a sequence of letters **F,B,L,R,U,D**, and we apply these operations in the given sequence. For example, if the cube is in the default state



then, by calling `applyMoves("RURRRUUU")` the state will change to



2) `public boolean isSolved()`

Returns true if the cube is in the solved state.

3) `public String toString()`

Returns the string in the same format as the file (including the '\n' characters).

For an example see the method `testReadFromFile1()` in `TestRubiksCube.java`.

4) `public static int order(String moves)`

Given a sequence of moves, the order of the sequence is defined to be the minimal integer $n > 0$ such that if we start from an initial state and apply the sequence exactly n times, the cube returns to the initial state. Note that this is a static method, and does not require a state of the cube.