IIP Second Partial - ETSInf
Date: January, 12th, 2015. Time: 2:30 hours.

1. 6.5 points It is available the `Block` class, that allows to represent blocks that can be stacked on towers for a game, already known, whose documentation is below:

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`BLUE` Constant which indicates that a Block is blue.
`static int`	`RED` Constant which indicates that a Block is red.

Constructor Summary

Constructors

Constructor and Description
`Block()` Creates a blue Block that is not wildcard, with random dimension in the range [1,50].
`Block(int color, int dimension, boolean wildcard)` Creates a Block with the given values for color, dimension, and wildcard.

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`canBeOn(Block b2)` Checks if current Block object can be on the given Block object; a Block b1 can be on a Block b2 if and only if the dimension of b1 is lower or equal than that of Block b2 and, it is a wildcard or colors for b1 and b2 are different.
`boolean`	`equals(java.lang.Object o)` Checks if current Block object is equal to the given Block object; i.e., have the same color and dimension and they both are wildcard or not.
`int`	`getColor()` Returns current Block object color.
`int`	`getDimension()` Returns current Block object dimension.
`boolean`	`getWildcard()` Checks if current Block object is a wildcard.
`java.lang.String`	`toString()` Returns a String with the current Block object data in a format similar to that of the following examples: "(Color: red, dimension: 22 and IT IS wildcard)", "(Color: blue, dimension: 15 and IT IS NOT wildcard)".

To stack up a block tower, the following rules are mandatory:

- Blocks stacked on a tower must follow alternate colors (on a blue block, only a red block can be stacked, and vice versa)
- On a block of dimension x, only a block of dimension y, with $y \leq x$, can be stacked (the tower becomes narrower towards the top, i.e., becomes wider towards the base)
- A block can be a **wildcard**; in that case, it can be stacked on a block of any color, although it must fulfil the dimension restriction

You must: implement the class `BlockTower` that represents a block tower by using the attributes and methods described below.

Remember that the constants defined in `Block` and `BlockTower` must be used whenever they are needed.

a) (0.5 points) Attributes:

- `MAX_BLOCKS`, class constant (static) that gives the maximum number of blocks for a tower (10)
- `numBlocks`, integer in the interval `[0,MAX_BLOCKS]` that represents the actual number of blocks of the tower in the current state
- `tower`, array of `Block`, with size `MAX_BLOCKS`. The elements of this array are stored sequentially according to the rules of the game, by using consecutive positions from 0 to `numBlocks-1`; thus, the base is `tower[0]` and the top is `tower[numBlocks-1]`
- `numWildcardBlocks`, represents the number of blocks that are wildcards in the current tower

b) (0.5 points) Default constructor (without parameters) that creates an empty tower (with 0 blocks)

c) (1 point) A method with header:

```
private int positionOf(Block b)
```

that, given a `Block b`, returns the position of the first occurrence of the block in the tower from the base, or -1 when it is not present

d) (1 point) A method with header:

```
public boolean push(Block b)
```

that returns `true` after stacking the block `b` on the tower; if there is no room for `b` or it cannot be on the top of the tower, the method must return `false` to signal that it could not be stacked; attribute `numWildcardBlocks` must be updated properly

e) (1 point) A method with header:

```
public Block firstGreaterThan(Block b)
```

that returns the first `Block` object of the tower, starting from the base, whose dimension is higher than that of the parameter `b`, or `null` when no block is greater

f) (1 point) A method with header:

```
public Block[] filterWildcardBlocks()
```

that returns an array of `Block` with the wildcard blocks of the tower; the length of this array it will be equal to the number of wildcard blocks, or 0 when no wildcard blocks are in the tower

g) (1.5 points) A method with header:

```
public String toString()
```

that returns "`Empty tower`" when the tower is empty or, otherwise, returns a `String` with a representation of the blocks that form the tower; for example, for a 5-block tower with these contents:

- `tower[0]`: red block, dimension 15, no wildcard.
- `tower[1]`: blue block, dimension 10, no wildcard.
- `tower[2]`: blue block, dimension 7, wildcard.
- `tower[3]`: red block, dimension 4, no wildcard.
- `tower[4]`: blue block, dimension 2, no wildcard.

The resulting `String` must be:

```
BB
RRRR
WWWWWWW
BBBBBBBBB
RRRRRRRRRRRRRR
```

where "`W`" indicates that is a wildcard block, "`B`" that is a blue block, and "`R`" that is a red block Notice that there are `numBlocks` lines and that in each line appear as many characters as the dimension of the represented block, with the characters indicating the color/wildcard property

Solution:

```
public class BlockTower {
    public static final int MAX_BLOCKS = 10;
    private Block[] tower;
    private int numBlocks, numWildcardBlocks;

    public BlockTower() {
       tower = new Block[MAX_BLOCKS];
```

```
        numBlocks = 0;
        numWildcardBlocks = 0;
    }

    private int positionOf(Block b) {
        int i = 0;
        while (i < numBlocks && !tower[i].equals(b)) i++;
        if (i < numBlocks) return i;
        else return -1;
    }

    public boolean push(Block b) {
        boolean res = false;
        if (numBlocks != MAX_BLOCKS
            && (numBlocks == 0 || b.canBeOn(tower[numBlocks-1]))) {
                tower[numBlocks++] = b;
                if (tower[numBlocks-1].getWildcard()) numWildcardBlocks++;
                res = true;
        }
        return res;
    }

    public Block firstGreaterThan(Block b) {
     return (numBlocks != 0 && tower[0].getDimension() > b.getDimension()) ? tower[0] : null;
    }

    public Block[] filterWildcardBlocks() {
        Block[] aux = new Block[numWildcardBlocks];
        for (int i = 0, k = 0; k < numWildcardBlocks; i++)
            if (tower[i].getWildcard()) {
                aux[k] = tower[i];
                k++;
            }
        return aux;
    }

    public String toString() {
        String res = "";
        for (int i = numBlocks - 1; i >= 0; i--) {
            String color = "R";
            if (tower[i].getWildcard()) color = "W";
            else if (tower[i].getColor() == Block.BLUE) color = "B";
            for (int j = 1; j <= tower[i].getDimension(); j++) res+=color;
            res += "\n";
        }
        return (numBlocks==0 ? "Empty tower" : res);
    }
}
```

2. 1.75 points A natural number is said to be perfect when it is equal to the sum of all its divisors, except itself. **You must:** Implement a class (static) method that checks whether an integer `n`, `n>0`, is perfect. For example, for `n` equal to `28`, the method must return `true` since its divisors are `1`, `2`, `4`, `7`, and `14`, whose sum is `28`.

Solution:

```
/** n > 0 */
public static boolean perfect(int n) {
    int sum = 1, i = 2;
    while (i <= n / 2) {
        if (n % i == 0) sum += i;
        i++;
    }
    return sum == n;
}
```

3. 1.75 points **You must:** Implement a class (static) method with parameters an array of integers `a` (`a.length` $>$ `0`) and an integer `p` which represents a valid position on the array (i.e., `0` $\leq$ `p` $<$ `a.length`). The method must return the maximum value of the sums of the elements of the array on the previous and posterior positions to the given position, without including it. For example, given the array `{1, 7, -2, 3, 4, 8, 1, -4}` and the position `2`, it will return the maximum between `1 + 7 = 8` and `3 + 4 + 8 + 1 - 4 = 12`, i.e., `12`.

Solution:

```
/** a.length > 0 and 0 <= p < a.length */
public static int maxSumPartition(int[] a, int p) {
    int sum1 = 0, sum2 = 0;
    for (int i = 0; i < p; i++) sum1 += a[i];
    for (int i = p + 1; i < a.length; i++) sum2 += a[i];
    if (sum1 > sum2) return sum1;
    else return sum2;
}
```