CS 129 HW 7 Block Tower

Write a program that builds a vertical tower out of blocks. There are three kinds of blocks:

- Stone – a gray block that weighs 20 units and has a load capacity of 100 units
- Brick – a red block that weights 10 units and has a load capacity 50 units
- Straw – a yellow block that weights 2 units and has a load capacity of 5 units

When the user hits the “a” key, it adds a block of stone. The “s” key adds a block made of brick, and the “d” key adds one made out of straw. New blocks get stacked at the top of the tower. The weight of all the blocks above a given block is its current load. If a block has exceeded its load capacity, it turns black, indicating the tower is not a valid configuration.

If a block hasn’t failed, show the current load it’s experiencing on the block.

This program is best achieved using inheritance. You must extend one at least one class for full credit. Good inheritance also means you never have to check what kind of object you have. So, to get full credit you cannot check type, or anything equivalent. Treat all blocks the same. For example, if you have any statements like these, you’ll lose a lot of credit:

- `if(block.type == straw)`
- `if(block.weight == 20)`
- `if(block.getType == "straw")`

Extra credit (10%) if the “f” key automatically generates the “ideal” tower: the tallest possible tower with no failures. You have to solve this with code and logic to get credit, you can’t just figure out the solution and set indexes manually.

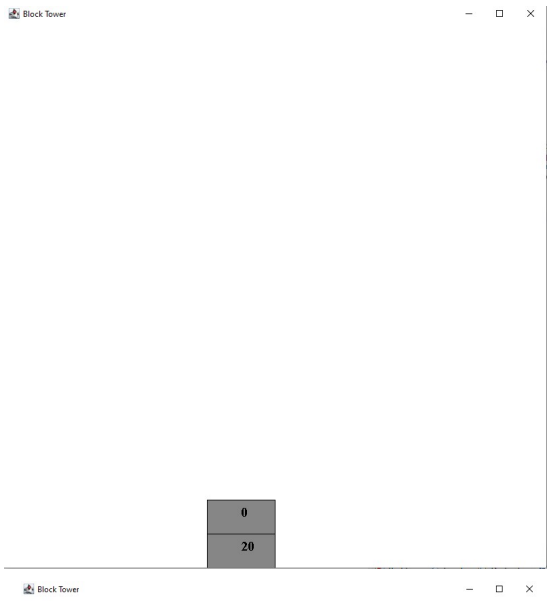
Extra credit (10%) if your “f” key implementation also works if the brick weights and load capacities change. You can assume stone will always have the highest values, then brick, then straw.

Up to 5% extra credit for exceptional documentation.

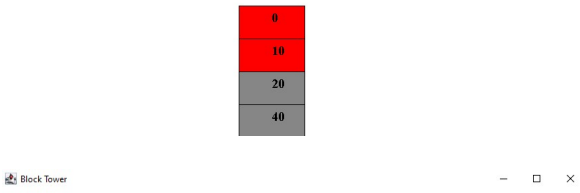
A starter for the tower is provided on the next page, you can add or remove code as necessary, or you can write your own entirely, it’s up to you. Some pictures are below the starter.

```java
public class Tower {

	Block[] blocks = new Block[40];
	int nextBlock = 0;

	/**
	 * Adds a block to the tower. If the tower is full, does nothing.
	 *
	 * @param b
	 */
	public void addBlock(Block b) {
		if (nextBlock > blocks.length - 1) {
			return;
		}
		blocks[nextBlock] = b;
		nextBlock++;
	}

	/**
	 * Checks the load capacity of all blocks in the tower against the blocks above
	 * them
	 */
	public void checkBlocks() {
		for (int i = 0; i < blocks.length; i++) {
			if (blocks[i] != null) {
				int totalWeight = 0;
				for (int k = i + 1; k < blocks.length; k++) {
					if (blocks[k] != null) {
						totalWeight += blocks[k].weight;
					}
				}
				blocks[i].checkFailed(totalWeight);
			}
		}
	}

	/**
	 * Draws all the blocks in the tower
	 *
	 * @param g
	 */
	public void draw(Graphics2D g) {
		for (int i = 0; i < blocks.length; i++) {
			if (blocks[i] != null) {
				blocks[i].draw(g);
			}
		}
	}

	/**
	 * Places the display elements of the tower's blocks using the given x and y
	 * location as the top-left corner of the bottom block. Note that index 0 of
	 * {@link #blocks} is the bottom, and the last index is the top
	 *
	 * @param xStart
	 * @param yStart
	 */
	public void placeBlockDisplays(double xStart, double yStart) {
		double y = yStart;
		for (int i = 0; i < blocks.length; i++) {
			if (blocks[i] != null) {
				blocks[i].setDisplayLocation(xStart, y);
				y -= Block.BLOCK_HEIGHT;
			}
		}
	}

}
```

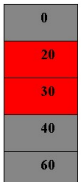
A couple stones to start (a key)



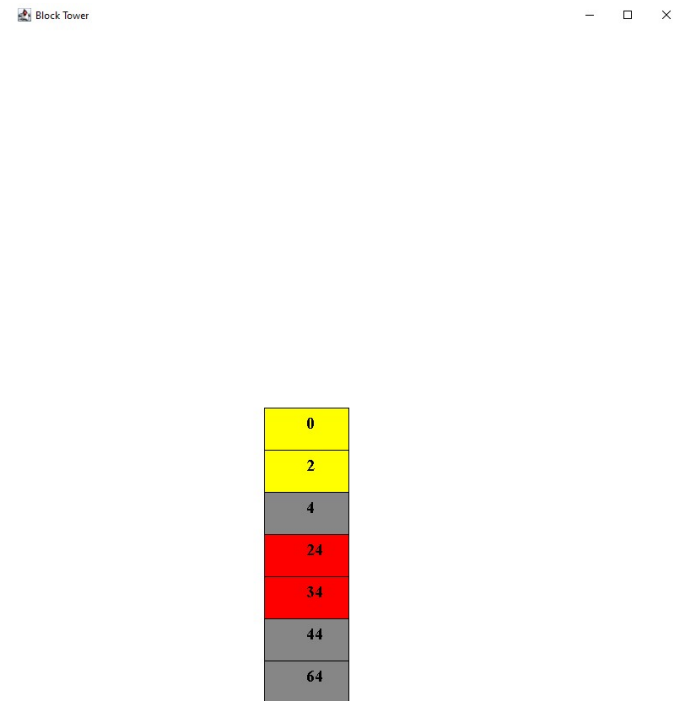
Then a couple bricks (s key)



And another stone.. (a key)



Finally, some straw on top (d key)



Try adding another stone or two (a key), and it’s a goner!