

1 Packages have arrived

In the following classes, cross out the lines that will result in an error (either during compilation or execution). Next to each crossed-out line write a replacement for the line that correctly carries out the evident intent of the erroneous line.

Each replacement must be a single statement. Change as few lines as possible.

After your corrections, what is printed from running **java P2.C5**?

```
package P1;
```

Write output here:

```
public class C1 {
```

```
    private int a = 1;
```

```
    protected int b = 2;
```

```
    int c = 3;
```

```
    public static int d() {
```

```
        return 13;
```

```
    }
```

```
    public void setA(int v) { a = v; }
```

```
    public void setB(int v) { b = v; }
```

```
    public void setC(int v) { c = v; }
```

```
    public int getA() { return a; }
```

```
    public int getB() { return b; }
```

```
    public int getC() { return c; }
```

```
    public String toString() {
```

```
        return a + " " + getB() + " " + getC() + " " + d();
```

```
    }
```

```
}
```

_____ 1 2 3 13 _____

_____ 20 60 40 13 _____

_____ 15 2 3 14 _____

```
package P1;
```

```
public class C2 extends C1 {
```

```
    public C2() {}
```

```
    public C2(int a, int b, int c) {
```

```
        setA(a);
```

```
        this.b = b;
```

```
        this.c = c;
```

```
    }
```

```
    public static int d() {
```

```
        return 14;
```

```
    }
```

```
    public C1 gen() {
```

```
        return new C3();
```

```
    }
```

```
}
```

```
package P1;

public class C3 extends C2 {
    private int a = 15;
    public String toString() {
        return a + " " + getB() + " " + getC() + " " + d();
    }
}
```

```
package P2;

class C4 extends P1.C2 {
    public int getB() {
        return 2 * b;
    }
    public C4(int a, int b, int c) {
        setA(a);
        this.b = b;
        setC(c);
    }
    public C4(int v) {
        super(v, v, v);
    }
}
```

```
package P2;
class C5 {
    public static void main(String... args) {
        P1.C1 x = new P1.C1();
        P1.C2 y = new C4(20, 30, 40);
        P1.C3 z = (P1.C3) y.gen();

        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
    }
}
```

2 Max Pooling

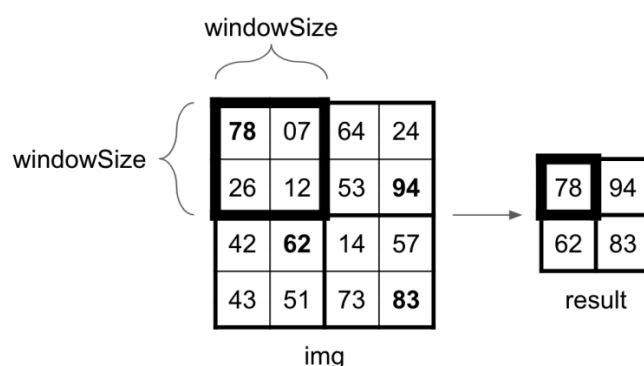
In this question, you will implement a function for performing max pooling, which is a technique used in deep learning for reducing the resolution of an image being fed through the layers of a neural network. (Summer 2019, MT1)

Images can be represented by two-dimensional arrays; the first dimension represents the rows of the image, and the second dimension is to represent the value of the pixel at a particular column index within a row.

For example, given a 2-dimensional image `int[][]` array called `img`, the pixel in the 5th row, and the 7th column can be accessed via `img[5][7]`.

One way we can implement max pooling is by cutting up our image (`img`) into small equal sized squares, and taking only the largest pixel value in each piece as the representative for that piece in our final downsampled image.

For example, this 4 by 4 `img` can be broken down into 4 equal-sized pieces of shape `windowSize` by `windowSize` (where `windowSize` is 2). The largest value in each of these pieces is recorded in result in their corresponding locations.



Fill in the blanks so that the function behaves as intended. (Hint: In Java, an `int` divided by an `int` will always result in a ...?)

```
static int[][] maxPool(int[][] img, int windowSize) {
    // resRows are the number of rows in result.
    // resCols are the number of columns in result.

    int resRows = img.length / windowSize;
    int resCols = img[0].length / windowSize;
    int[][] result = new int[resRows][resCols];

    for (int r = 0; r < img.length ; r++ ) {
        for (int c = 0; c < img[0].length ; c++ ) {
            // Java's Math.max() function only accepts two arguments at a
            // time.
            // (Put one on the first line, and the second on the line below
            // it).

            int largestSoFar = Math.max(img[r][c],
                                        result[r/windowSize][c/windowSize]);
            result[r/windowSize][c/windowSize] = largestSoFar;
        }
    }

    return result;
}
```

```
}
```

3 Iterator of Iterators

Implement an `IteratorOfIterators` which will accept as an argument a `List` of `Iterator` objects containing `Integers`. The first call to `next()` should return the first item from the first iterator in the list. The second call to `next()` should return the first item from the second iterator in the list. If the list contained n iterators, the $n+1$ th time that we call `next()`, we would return the second item of the first iterator in the list.

For example, if we had 3 `Iterators` A, B, and C such that A contained the values [1, 2, 3], B contained the values [4, 5, 6], and C contained the values [7, 8, 9], calls to `next()` for our `IteratorOfIterators` would return [1, 4, 7, 2, 5, 8, 3, 6, 9].

Feel free to modify the input `a` as needed.

Note - this is only one possible solution, as there are many others.

```
import java.util.*;
public class IteratorOfIterators implements Iterator<Integer> {
    LinkedList<Integer> l;
    public IteratorOfIterators (ArrayList<Iterator<Integer>> a) {
        l = new LinkedList<>();
        int i = 0;
        while (!a.isEmpty()) {
            Iterator<Integer> curr = a.get(i);
            if (!curr.hasNext()) {
                a.remove(curr);
                i -= 1; //or else we'll skip an element
            } else {
                l.add(curr.next());
            }
            if (a.isEmpty()) { //could've removed the last Iterator
                break;
            }
            i = (i + 1) % a.size();
        }
    }

    @Override
    public boolean hasNext() {
        return !l.isEmpty();
    }

    @Override
    public Integer next() {
        return l.removeFirst();
    }
}
```