

**Problem 1 (3 points)**

While working on his solution to project 2, Mike Clancy encountered an interesting bug. His program includes a `LineNumber` class that supplies, among other methods, a constructor that builds a `LineNumber` object from a `String`. At the end of each call to `extendProof`, he stores the current line number and expression in a `HashMap`, then increments the line number and returns.

One of Mike's early tests cases involved the following code, which produced the error message about something being really wrong:

```
HashMap lineValues = new HashMap ( );  
...  
lineValues.put (new LineNumber ("1.1"), new Expression ("(a|b)"));  
Expression expr = lineValues.get (new LineNumber ("1.1"));  
if (expr == null) {  
    System.err.println ("something is really wrong here");  
}
```

The problem turned out to be an incorrect parameter declaration in a method. Identify the method, describe why the error would produce the indicated behavior, and supply the correct method header.

**Problem 2 (6 points)**

*Part a*

Describe, in terms that another CS 61BL student can easily translate into code, a data structure for songs named `songsByGenre` that would enable iteration of all songs of a given genre to be as fast as possible. Use `java.util` classes wherever possible. Assume that songs are represented by the `Song` class from project 1, which provides a constructor and the following methods:

```
public CD cd ( );  
public String title ( );  
public String artist ( );  
public String genre ( );  
public Time time ( );
```


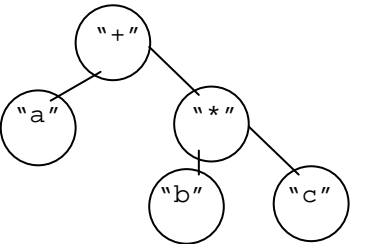
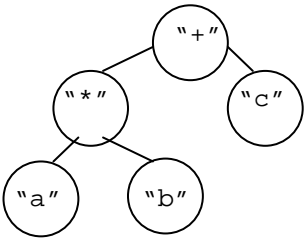
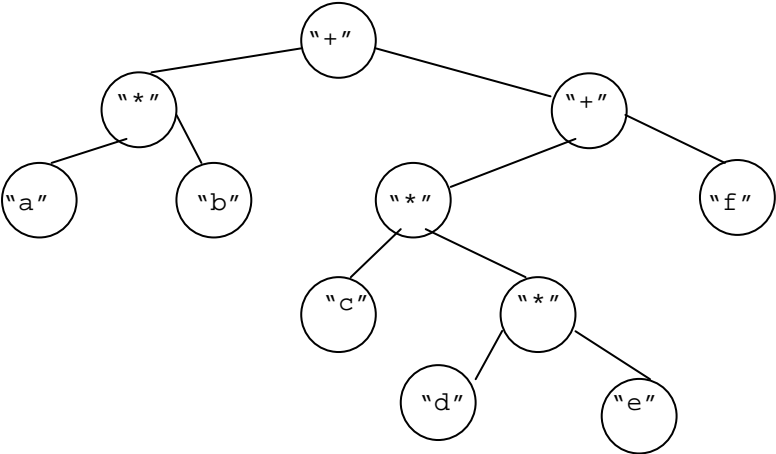
*Part b*

Write the `put` method that adds a `Song` object to your data structure. This problem will be graded only if your answer to *part a* allows sufficiently fast iteration by genre.

**Problem 3 (7 points)**

Write the `BinaryTree.noParensArithExpTree` method on the next page. It is given as argument a `String` representing an arithmetic expression, and returns a corresponding expression tree. The `String` argument will be nonempty and will consist only of the characters '\*', '+', and lower case letters (you don't need to verify this). Uses of + and \* should associate to the right; e.g. the expression "a\*b\*c" means (a\*(b\*c)).

In translating the expression to a tree, you should give multiplication higher precedence than addition as you do in conventional algebraic notation. Some examples:

<i>expression</i>	a	a+b*c	a*b+c
<i>trees</i>			
<i>expression</i>	a*b+c*d*e+f		
<i>tree</i>			

You may use helper methods, but don't add any instance variables. Relevant methods in the `String` class are the following.

- `indexOf` returns the position in the string of its character argument or -1 if the character does not appear in the string.
- `substring`, given two `int` arguments `beginIndex` and `endIndex`, returns the substring that starts at position `beginIndex` and ends at `endIndex-1`. With only one position argument, it returns the substring from the given position to the end of the string.

**Your answer to problem 3**

```
// Return a binary expression tree that represents the given expression string, with  
// * having higher precedence than + and consecutive uses of * or + associating to the right.  
// Precondition: the expression is nonempty and contains only *, +, and single-character  
// variable names.
```

```
public BinaryTree noParensArithExprTree (String expr) {
```

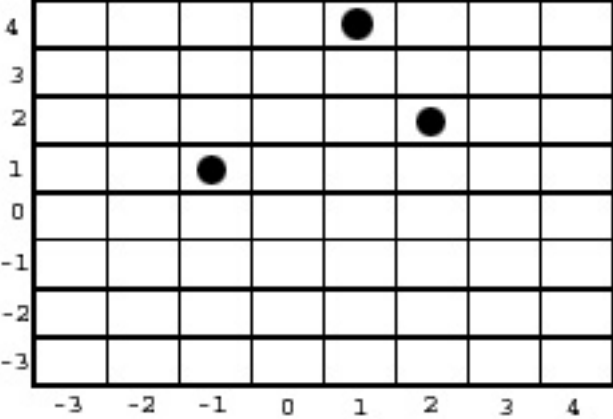
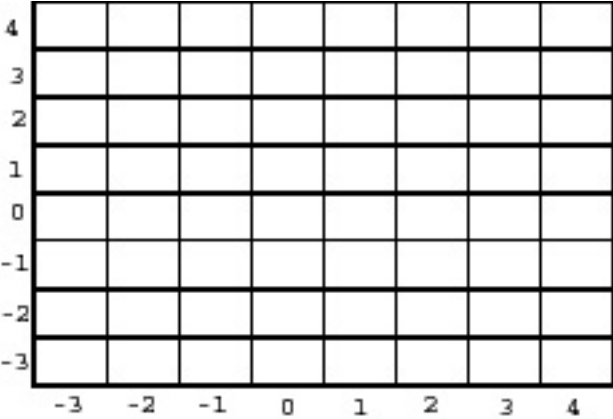
**Problem 4 (7 points)**

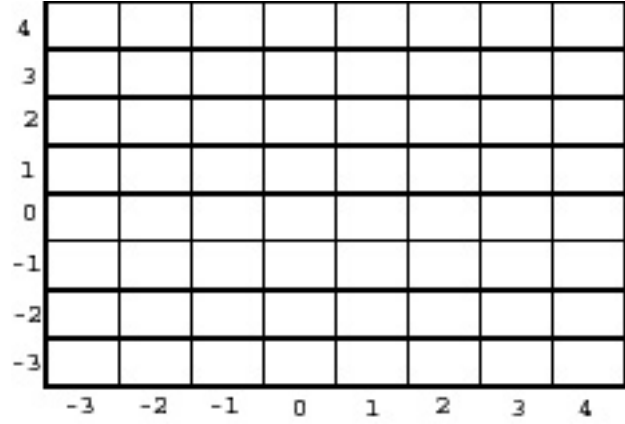
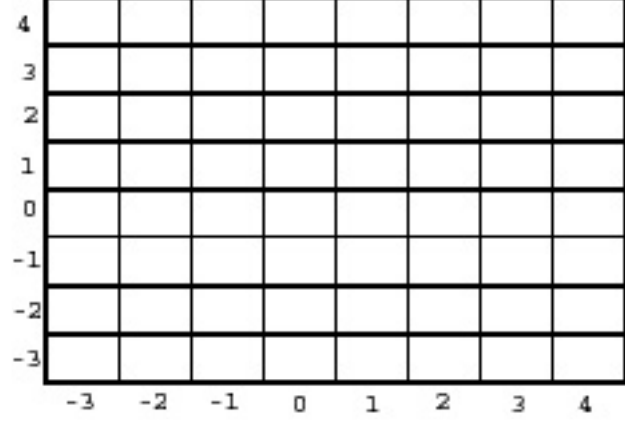
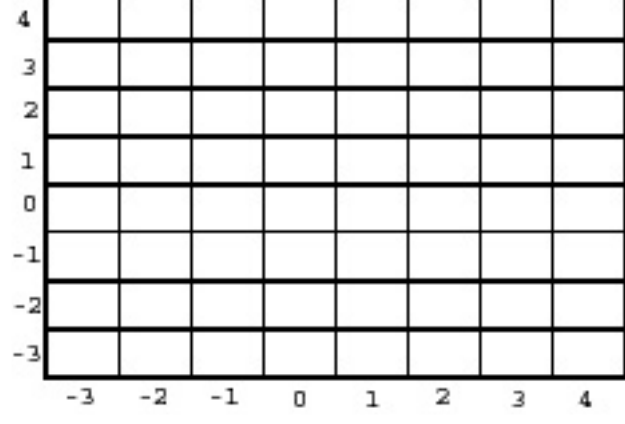
Suppose you were required to add a `remove` method to the `QuadTree` class you complete in lab (with the `contains` method and a full node detector). It would be declared as

```
// Remove p from the set of points represented by this quad tree.
// Precondition: p is a member of the set of points represented by this quad tree.

void remove (Pt p) {
    ...
}
```

Below, give four additional points as test arguments to `remove` that collectively would provide you the most evidence of the correctness of the `remove` method. Also describe the evidence of correctness they would provide. An example test call is provided. Unlike in the example, your calls should all assume that the point to be deleted is somewhere in the tree.

<i>quad tree situation</i>	<i>point to be deleted</i>	<i>rationale for test</i>
	$(-1, 0)$	check what happens when the precondition isn't satisfied
		

<i>quad tree situation</i>	<i>point to be deleted</i>	<i>rationale for test</i>
		
		
		

**Problem 5 (7 points)**

*Part a*

In terms that another CS 61BL student can easily translate into Java code, provide a pseudocode implementation of the following method for the `Heap` class:

```
// Remove the item at position k in the myValues vector;  
// myValues represents a legal binary max heap of the remaining elements  
// upon return from delete.  
// Precondition: 0 ≤ k < myValues.size ( )  
  
void delete (int k) {  
    . . .  
}
```

You may refer to methods in *Data Structures into Java* or in the Course Portal without describing them further. Your method should run as fast as possible in the big-Oh sense.

*Part b*

Give a big-Oh estimate that's as specific as possible for the number of comparisons your method needs to delete the `k`th item in the heap, along with a brief explanation of your answer.