CSE<8B> Today

Reminders:
Monday: NO CLASS
Wednesday: Exam 4

Connect 4 Player:
PSA and YouTube videos posted
OPTIONAL, due Friday week 10

PSA 7 due this Saturday
(but limited tutor hours over the weekend)

Exam 4 reviews:
Today’s discussion
Tuesday during my office hours

PSA 8: out by next Tuesday
(due Thursday, week 10)
1. What does a generic method or class allow you to specify?

A. Whether it is static or not.

B. Who can use the method or class.

C. Types of objects that the class or method may work with.

D. The type of the class itself or the return type of the method.
2. Write a line of Java code to declare and instantiate an ArrayList of (only) Character objects.

A. `ArrayList<T> a = new ArrayList<T>();`

B. `ArrayList a = new ArrayList();`

C. `ArrayList<Object> a = new ArrayList<Object>();`

D. `ArrayList<Character> a = new ArrayList<Character>();`

(D) is the correct answer.
3. Can you define a generic type to be a primitive type?

A. Yes

B. No

```java
class Example {
    public int realFind(String[] myList, String toFind) {
        return find(myList, toFind, 0);
    }

    int find(String[] myList, String toFind, int index) {
        // Implementation...
    }
}
```
public int find( String[] myList, String toFind, int currIndex )
{
    currIndex = 0;
    if ( currIndex >= myList.length )
        return -1; // It wasn’t there!
    if ( myList[currIndex].equals( toFind ) )
        return currIndex; // Found it!
    else {
        [what goes here??] ________

    }
    return 0;
}

A. return find( myList, toFind, currIndex + 1 );
B. return find( myList, toFind, currIndex - 1 );
C. return currIndex + 1; - no rec. call
D. find( myList, toFind, currIndex + 1 );
E. None of these.

Why does currIndex have to be a parameter?

Explain why each answer is wrong or right!
Worksheet: Complete the recursive step

A. `return find( myList, toFind, currIndex + 1 );`
This is the correct answer. It will reduce the problem by increasing `currIndex` until it hits the base case. It will also return the value of the recursive call.

B. `return find( myList, toFind, currIndex - 1 );`
This answer reduces the problem, but does so in a way that takes it away from the base case, instead of towards it.

C. `return currIndex + 1;`
This answer does not make a recursive call. So the method will fail to look in the rest of the list for the element.

D. `find( myList, toFind, currIndex + 1 );`
This answer makes the correct recursive call, but does not return its result. We must not only make the recursive call, but also return its result.

E. None of these. Wrong because A is correct.
Ever used a telephone book?

• If I want to look up someone’s number and their last name starts with Z, where should I start?
• If I want to look up someone’s number and their last name starts with G where should I start?
Binary Search
(only works for “ordered” data)

• Look in the middle of the data list (array).
  – If that element is what you are looking for – return it’s index
  – If not
    • If what you are looking for “comes before” in the ordering, look in the half on the left
    • Else (what you are looking for “comes after” in the ordering), look in the half on the right
What is the correct method header for binarySearch? (What data do you need to keep track of as you search?) The method will return the *position* of the item if it is found.

- A. `public int binarySearch(ArrayList theList, int toFind)`  
- B. `public int binarySearch(ArrayList theList, int toFind, int middle)`  
- C. `public int binarySearch(ArrayList theList, int toFind, int low, int high)`  

*Superset of information from A & B*
Binary Search example

How can we take advantage of knowing that we are searching for an int??

A. public int binarySearch( ArrayList<int> theList, int toFind, int low, int high)
B. public int binarySearch( ArrayList<Integer> theList, int toFind, int low, int high)
C. public int binarySearch( ArrayList<int> theList, int toFind, int low, int high)
D. public int binarySearch( ArrayList<Integer> theList, Integer toFind, int low, int high)

Syntax issue.
Algorithm “pseudo-code”:
1. calculate the midpoint, mid, between low and high
2. If theList[mid] == toFind, return mid
3. if toFind is larger than theList[mid], recurse on the larger half of the list
4. else if toFind is smaller than theList[mid], recurse on the smaller half of the list

Trace the values of high, low and mid when you call binarySearch with this list above and the value 8. Draw more stack frames if you need them.
public int binarySearch( ArrayList<Integer> list,  
                 int toFind, int low, int high ) {  
    int mid = (low+high) / 2;  
    if (toFind < list.get( mid ))  
        return ________________________________  
    else if (toFind == list.get( mid ))  
        return ________________________________  
    else // toFind is > list.get(mid)  
        return ________________________________  
}
public int binarySearch( ArrayList<Integer> list, 
                      int toFind, int low, int high ) {

    int mid = (low+high) / 2;
    if (toFind < list.get(mid))
        return binarySearch( list, toFind, low, mid-1 );
    else if (toFind == list.get(mid))
        return mid;
    else // toFind is > list.get(mid)
        return binarySearch( list, toFind, mid+1, high );
}

Which is the base case in the method above?
public int binarySearch( ArrayList<Integer> list, int toFind, int low, int high ) {

    int mid = (low+high) / 2;
    if (toFind < list.get( mid ))

        return binarySearch( list, toFind, low, mid-1 );

    else if (toFind == list.get( mid ))

        return mid;

    else // toFind is > list.get(mid)

        return binarySearch( list, toFind, mid+1, high );
}

When will this method not work?
A. When the element you are looking for is the first element in the list
B. When the element you are looking for is the last element in the list
C. When the element you are looking for is not in the list
D. When the list is not sorted.
E. More than one of these.