Advanced Programming Techniques in Java

ArrayList

Lecture 17

Class Objectives

- `ArrayList` (section 10.1)

- Errors (Section 1.3)
- Complexity & Running time (section 13.2)

Review: ArrayList of any

- When constructing an `ArrayList`, you must specif between `< >`
- By making the `ArrayList` class a Generic class, the sa different types

- **Syntax**: `ArrayList<Type> name = new ArrayLis`

```
ArrayList<String> names = new ArrayList<St
```

- Java 7's shorter "diamond operator" syntax

```
ArrayList<String> names = new ArrayList
```

Review: `ArrayList` of any typ (cont.)

- You can store any type of object in an `ArrayList`
- `ArrayList<Point> points = new ArrayList<Po`
- The `points` list will manipulate and return `Points`
- `ArrayList<Color> points = new ArrayList<Co`
- The `points` list will manipulate and return `Colors`

Review: Issues with dynamic a

- Assume you have an `ArrayListwords words = ago]`
  - You want to add '~' before each word

```
for (int i=0; i < words.size()
    words.add(i,'~');
}
```

- Does this work? - NO!

Review: Issues with dynamic remo

- We now want to redo this operation (remove '~')
- Write code that removes every other element startin

```
four, ~, score, ~, and, ~,seven, ~, years,
```

- Does this work? Why?

```
for (int i=0; i < words.size();
    words.remove(i);
}
```

- Output `words = [four, ~, score, ~, and, ~,s = [four, ~, ~, and, ~,seven, ~, years, ~, and, seven, ~, years, ~, ago] ...`

Review: Solution 1

- Again, dynamic shifting causes the problem
- Once you remove an element, all the rest are shifted to th

Correct solution:

```
for (int i=0; i < words.size(); i++) { wor
}
```

- Output `words = [four, score, and, seven, ye ago]`

“Backwards” solution 1

```
for (int i=words.size()-2; i>=0; i-=2) {
        words.remove(i);
}
```

- Output

```
words = [~, four, ~, score, ~, and, ~,seven, ~, ye
words = [~, four, ~, score, ~, and, ~,seven, years
...
words = [four, score, and, seven, years, ago]
```

Accessing items

- To get an element at a specific index you can use t

```
int sum=0;
  for (int i=0; i < list.siz
                  i++) {
```

```
String s = list.get(i); sum
s.length();
}
```

Avoid expensive shifts

- Every time you call the `add` and `remove` method y
  array
  - That's expensive!

- You can use **`set(int index, E value)`**
- It replaces the element at the `index` position with the `va`

- **Example:** `list.set(0, "Harvard");`

- This replaces the first element in the list

Problem

- Write a program that reads a file and displays the w
  - **First display all words**
  - Then display them with all plurals capitalized
  - Then display them in reverse order
  - **Then display them with all plural words removed**

Solution v. 1

```
ArrayList<String> allWords = new Array
Scanner input = new Scanner(new File("
while (input.hasNext()) {
    String word = input.next();
    allWords.add(word);
}
System.out.println(allWords);

// Remove all plural words
for (int i = 0; i < allWords.size(); i
    String word = allWords.get(i); if
    (word.endsWith("s")) {
    allWords.remove(i); i--;
    }
```

```
}
```

Searching for elements

- You can search the list for an element:

```
if (list.contains("Chemistry")) {
    System.out.println("Chemistry is
} else {
    System.out.println("Chemistry is
}
```

- Output: `Chemistry is not found.`

- **`contains(Object o)`** Returns true if the list con

Where is my element?

- Sometimes you may need to know where a value is

- Example: you want to replace the first occurrence o
- You must tell the `set` method where the word to be repla
- Use **`indexOf(Object o)`**

The content of the lis
need to create a new

```
public static void replace (ArrayList<String> list,
          int index = list.indexOf(target); if
(index>=0) { list.set(index, replacement); } }
```

How `contains` and `indexOf`

- How do search methods `contains`, `indexOf` unc
  are looking for?
- Remember: objects are just references to a memory addr
  themselves!
- They are using the `equals` method

Out-of-bounds

- Legal indexes are between **0** and the **list's size() - 1**
- Reading or writing any index outside this range will cause

index	0	1	2	3	value
	Marty	Kevin	Vicki	Larry	

```
ArrayList<String> names = new ArrayList<Str
names.add("Marty");    names.add("Kevin");
names.add("Vicki");    names.add("Larry");
System.out.println(names.get(0));          //
System.out.println(names.get(3));          //
System.out.println(names.get(-1));         //
names.add(9, "Aimee");                     //
```

ArrayList methods

`add(`**value**`)`	appends value at en
`add(`**index**`, `**value**`)`	inserts given value shifting subsequent

`clear()`	removes all elemen
`indexOf(`**value**`)`	returns first index w (1 if not found)
`get(`**index**`)`	returns the value at
`remove(`**index**`)`	removes/returns va subsequent values
`set(`**index**`, `**value**`)`	replaces value at gi
`size()`	returns the number
`toString()`	returns a string repr as "`[3, 42, -7,`

`ArrayList` methods (cont.)

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from th (at the end of the list, or
`contains(`**value**`)`	returns true if given value
`containsAll(`**list**`)`	returns true if this list cor
`equals(`**list**`)`	returns true if given othe
`iterator()` `listIterator()`	returns an object used to later)
`lastIndexOf(`**value**`)`	returns last index value is
`remove(`**value**`)`	finds and removes the giv
`removeAll(`**list**`)`	removes any elements fo
`retainAll(`**list**`)`	removes any elements no

`subList(`**from**`, `**to**`)`	returns the sub-portion o indexes **from** (inclusive)
`toArray()`	returns the elements in th

Arrays vs. ArrayList

- Construction

```
String[] names = new String[5];
ArrayList<String> list = new ArrayList<St
```

- Storing a value `names[0] = "Jessica"; list.se`

```
list.add("Jessica");
```

- Retrieving a value

```
String s = names[0];
String s = list.get(0);
```

Arrays vs. ArrayList (co

- Doing something to each value that starts with "B"

```
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("B")) { ... }
}

for (int i = 0; i < list.size(); i++) { if
    (list.get(i).startsWith("B")) { ... }
}
```

- Seeing whether the value "Benson" is found `if (n`

```
if (list.contains("Benson")) { ... }
```

Use `contains` to eliminate dup

- Assume you have the following file, and you want to

names
- *Maria Derek Erica Livia Jack Anita Kendall Maria Li*

- Output `list = [Maria, Derek, Erica, Livia, Jack`

  `Jamie]`

Use `contains` to eliminate dup

- Assume you have the following file, and you want to
  names
  - *Maria Derek Erica Livia Jack Anita Kendall Maria Li*

```
Scanner input = new Scanner(new File
ArrayList<String> list = new ArrayLi
while (input.hasNext()){
            String name = input.next()
            if (!list.contains(name))
            list.add(name);
            }
}
System.out.println("list = " + list)
```

- Output `list = [Maria, Derek, Erica, Livia, Jack`

  `Jamie]`

`ArrayList` and enhanced for lo

- New loop syntax:

```
for (<type> <name> : <collection>) {
    <statement(s)>;
}
```

- This syntax can be used to examine an `ArrayLis`

```
int sum = 0; for (String s : list) { su
+= s.length();
}
System.out.println("Total of lengths = "
```

Easier but trickier!

- You cannot skip elements with this syntax
- You cannot modify the lists while you are iterating

```
ArrayList<String> list = new ArrayList<St
for (String s : list) {
    System.out.println(s);
    list.remove(0);
}
```

- **ConcurrentModificationException** error

Errors

Three main categories of

- Syntax Errors
- Run-time errors
- Logic errors

Syntax Errors

- You have a typo somewhere or wrot
  thecompiler didn't understand
- Easy to find because you just need t
  compileyour code

- Syntax errors are UNACCEPTABLE younever even had the chance to ru
- The compiler actually tells you what' code
- If you are overwhelmed by a multitud justlook at them one at a time (top-m compile again

Run-time Errors

- Run-time errors: Your program cra execution
- Reasonably easy to find with thorc testing,though **much** harder if cod

multithreaded and the error is the
condition

- Trace back where it happens to fig
wrong with the code; debuggers c
helpful with this type of error

Logic Errors

- Hardest ones to fix
- Program doesn't crash but produ
it doesn’t do what you intended

- May result in code that leads to conditionand manifests itself as somewhere else
- Hardest part is that the bug usua fromyour thought processes, ma wrong line of code is producing it isn't

Purposes ofTesting

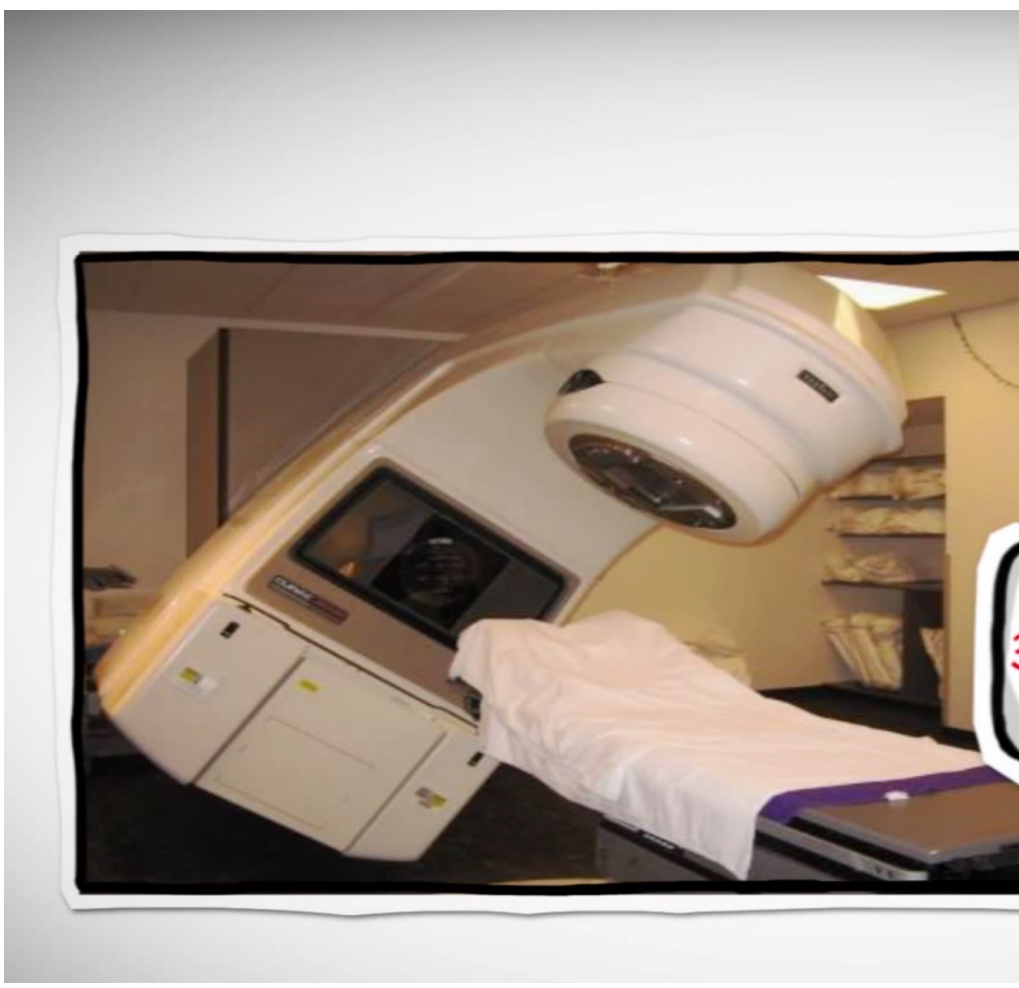
- To make sure the software
  - meets the requirements that guided i development,
  - responds correctly to all kinds of inpu

– performs its functions within an accep
– is sufficiently usable,
– can be installed and run in its intende and
– achieves the general result its stakeh

Why is Testing Importan

China Airlines Flight140, April 26

Why is Testing Importan



Why is Testing Important?

April, 1999

Why is Testing Importan

May, 1996

of modules contain most of t

- With experience, you can i modules
- However, this can lead to a

Pesticide Paradox

- If the same tests are repeated eventually the same test cases new bugs
- This is called the Pesticide Para
- To overcome this, the test case regularly reviewed and revised different test cases to help

- You can never claim that FREE
- Why?
- Absence of error is a fallacy

7 Testing Principles

- Testing shows presence of defe
- Exhaustive Testing is impossi
- Early Testing
- DefectClustering

- Pesticide Paradox
- Testing is context dependent
- Absence of errors is a fallacy

Data Structure

Runtime Eff

Data structure

- a way to store and organize data in or
  and modification.
- no single data structure optimal for all
- usually optimized for a specific proble
- important to know the strength and lim
  them.

Examples:

- Trees (binary search trees, red-black
- Stacks (last in, first out), queues (first
  queues.

Algorithms

Algorithms:

- well-defined computational procedure
- takes value or set of values as input.
- produces value or set of values as ou
- tool for solving a **well-specified** comp
  instance of a problem consists of t
  compute a solution to the problem.
- correct algorithm solves the given cor

Sorting problem:

Input: A sequence of n numbers $a_1,...,a$

Output: A permutation $a_1,...,a_n$ of the inpu
that $a_1 \leq \cdots \leq a_n$.

Efficiency

Computing time and memory are bounded resource

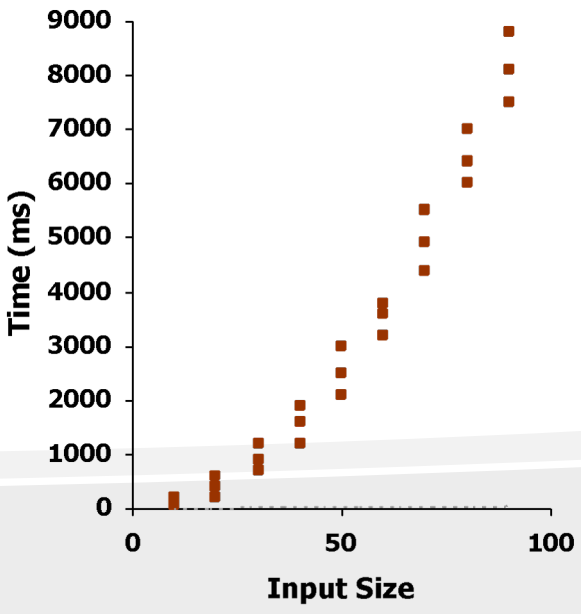
Efficiency:

- Different algorithms that solve the sar
in their efficiency.
- More significant than differences due
memory, disks, ...) and software (OS,
compiler, ...).

=> Running Time/Computational Complexity

Empirical Analysis

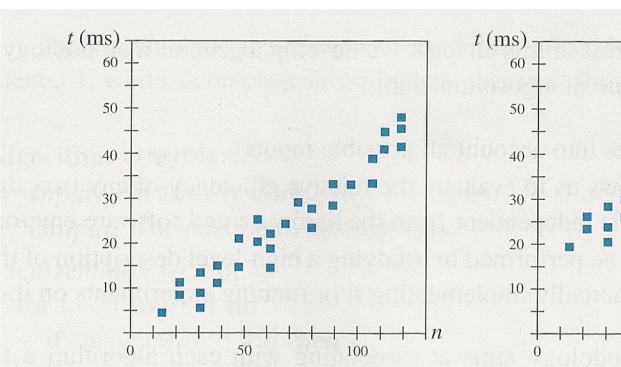
- Run time can be studied experimentally
  - Write a program implementing the algorithm
  - Run the program with inputs of varying size
  - Get an accurate measure of the actual running time



- Plot the results

Limitations of Empirical Analysis

- Experiment can be done only on a limited set of test i
- Difficult to compare the efficiency of two algorithms u on same environment
  - Hardware environment (processor, clock, rate, memory
  - Software environment (OS, programming language, co



- Necessary to implement and execute an algorithm to