Study guide
CSCI 2120 Midterm 2
Fall 2015

Topics:

Generics

1. What does generic mean?

It can use any type. Array

```
public T getAtIndex(int index){
	return array[index];
}
```

T is the generic type. If we had a String array, the return type is String. If we had an array of arrays, the return type is array.

If we're making a class with a generic type

```
public class ClassName<T>{
	private T[] array;
```

if I make an object of type ClassName with <String>, the type of the array is String. If I make an object of type ClassName with <BigInteger>, the type of the array is BigInteger.

2. What advantages do we get from using generics?

Reusable code. Don't have to rewrite everything.

3. What are type bounds on generics?

Type bounds restrictions on what type of generics we can use.

```
public class ClassName<T extends Number>{

	public T add(T one, T two){
		....
	}
}

public class ClassName<T extends Comparable>{

	private T data;
	private T[] array;

	public int compareTo(T other){
		return data.compareTo(other);
	}

	public T getGreatest(){
```

```
            ... compareTo to find teh greatest
        }
}
```

4. Is it possible to use primitive types in generic methods?

Yes

Wrapper classes:

boolean	Boolean
int	Integer
char	Character
float	Float
etc	

We have to specify the Wrapper class in creation.

ArrayList<Integer> intArray

LinkedList<Character> charArray

ArrayList<Boolean> boolArray

Autoboxing and autounboxing, Java can handle direct use of primitives.

```
intArray.add(5);                auto-boxing
charArray.add('a');             auto-boxing

intArray.get(1) + intArray.get(2)           auto-unboxing
if( boolArray.get(0) )                      auto-unboxing
```

5. What does raw type mean?

Raw type is not specifying a specific type.

```
Stack<File> fileStack
Stack<> thingStack
```

Advantage: useful if you're using things of several different types.
Disadvantage: Can't do anything specific on them. Dangerous for unchecked conversions or unsafe operations.

6. What are wild cards?

?

Wild cards have less restriction than regular generics.

public class Something<T extends Comparable>

public class Something<? extends Comparable>

7. Examples in code of generics vs static type.

```
String[] data

public boolean exists(String in){
	for (int i = 0; i < data.length; i++){
		if in.equals(data[i]) return true;
	}
	return false;
}

--------

T[] data

public boolean exists(T in){
	for (int i = 0; i < data.length; i++){
		if in.equals(data[i]) return true;
	}
	return false;
}
```

Generic Collections

1. What sort of collections do we have?

ArrayList, LinkedList, BinaryTree, Map, Stack, Queues, Array, Tree, Set

2. Is there anything special in any of the collections that differ from using static type and generics?

Java doesn't allow creation of generic arrays.

```
		// reflection
		data = (T[]) Array.newInstance(clazz, 0);
		// below would be with casting
	//	data = (T[]) new Object[0];
```

3. Examples in code of generics vs static type in collections

Lists & Collection Methods

1. What kind of lists do we have?

ArrayList, LinkedList

2. What's their structure?

ArrayList is an array, it's sequential. LinkedList has references to next and previous, and it uses Nodes.

Doubly linked list has two references, the next and previous nodes. Singly linked list has one

reference, usually the next node.

3. How do they differ from one another, in structure, advantages/disadvantages, etc.

Advantage of arrays/ArrayList: faster searching, indexing

Advantage of Linked List: easier to remove/add

4. What does “self-referential” mean? Give an example, too.

Self-referential means that it refers to other things of its type

```
private class Node
        Node previous
        Node next
        T data
```

5. What's an Iterator? What's its methods?

Iterator goes through a collection and it saves its spot.

hasNext() next() remove()

Collection c

Iterator i

// print out everything

```
while (i.hasNext()){
        System.out.print(next());
}
```

6. Examples in code

Generic Collections: Sets & Maps

1. What are sets and maps?

Sets are collections that can't have duplicates.

If we have 1, 23, 52, 3, 5, 3, 12, 3495 and we insert them into a list, we get 8 items. In a set would have 7 items, because there's duplicates (3).

Maps map a key to a value.

Keys: 2 our search algorithm uses mod 5

2. What are their required elements?

Sets can't have duplicates. Maps need keys and values.

3. What are the uses of each?
4. Examples in code

Searching, Sorting, and Big O: Linear & Binary Search

1. What is Big O notation? What does it mean?

Big O is worst case scenario of cost complexity.

2. What does Big O notation look like?

O(something)

3. List some examples of Big O notation, in order of their efficiency.

O(1) constant. (In Big O notation, all constant numbers are reduced to 1)
O(log n) logarithmic (e.g. Binary search)
O(n) linear
O(n log n)
O(n^2) quadratic
O(n^3)

4. List and describe the different types of searching algorithms

linear search, binary search

Linear search: goes through the whole collection. O(n)

Binary search: (uses ordered collection) starts in the middle, and halves it each time. O(log n)

0 3 6 9 12 15 18 21

Binary search for 2. Start in the middle at 12. Is this it? No. Is it greater or lesser? Lesser, so we go half again on the lower half. 0 3 6 9 start at the middle 3. Lesser, so go lower half again. 0. Is that it? no.

Linear search:

```
public boolean exists(T in){
	for (int i = 0; i < data.length; i++){
		if in.equals(data[i]) return true;
	}
	return false;
}
```

Binary search would use indices

5. List and describe the different types of sorting algorithms

Generic Classes and Methods

1. Examples

Custom Generic Data Structures (Lists)

1.

Custom Generic Data Structures (Stacks & Queues)

1. How are stacks and queues different from each other?

Stacks and queues are both ordered list data structures, but stacks are LIFO (last in, first out) and queues are FIFO (first in, first out). To remember this, you can think of a stack like a stack of plates – you would remove from the top, or the last recently placed item; a queue would be like a line of people at a store, where the first person in line will go next.

2. What are the uses of each?

The most familiar stack would be the execution stack (remember assembly). Queues are good for anything that needs a sequential, time-ordered execution.

3. We have methods add, remove, and look at the next item to be removed. What are the special names in stacks vs queues?

Stacks have push (add), pop (remove – last item), and peek (look at last item). Queues sometimes use the names enqueue (add), dequeue (remove – first item), and peek (look at first item).

4. What different types of queues do we have?

Custom Generic Data Structures (Trees & Binary Search)

1. What are trees?

Trees are data structures which are composed of nodes that have children. Typically trees will have a root node, and each node will have at least child nodes, sometimes a reference to its parent. A node that has no children is called a leaf. Usually nodes will have their child references set to null until a child is inserted.

2. What are some examples of different types of trees?

While some trees can have multiple children at each node, trees whose nodes can have at most a right and left child are called binary trees. Binary trees that are ordered with comparisons are called binary search trees.

3. List the 4 main methods of tree traversal, and describe in pseudocode

Breadth first; pre-order, in-order, post-order.

Breadth first is by level.

String result
queue<nodes> Q
Q.enqueue root

```
while Q isn't empty
        node temp = Q at 0
        if left child of temp isn't null
                Q.enqueue left child
        if right child of temp isn't null
                Q.enqueue right child
        result = result concatenate temp
        Q.dequeue
return result
```

Pre-order traversal is: N, L, R (this node, left child, right child). In-order traversal is L, N, R (left child, this node, right child). Post-order traversal is L, R, N (left child, right child, this node).

The order traversals are very easy to implement recursively, and the methods only have to be altered slightly between each.

```
preOrderTraverse( node )
print node                                      // this node
if left child isn't null
        return preOrderTraverse( left child )   // left child
if right child isn't null
        return preOrderTraverse( right child )  // right child

...

inOrderTraverse( node )
if left child isn't null
        return preOrderTraverse( left child )   // left child
print node                                      // this node
if right child isn't null
        return preOrderTraverse( right child )  // right child

…

postOrderTraverse( node )
if left child isn't null
        return preOrderTraverse( left child )   // left child
if right child isn't null
        return preOrderTraverse( right child )  // right child
print node                                      // this node
```

4. Code examples

More Design Patterns

Strings and Pattern Matching

1. What are strings in memory?

Strings are arrays of characters in memory.

2. How do we get the size of a string?

String s
s.length()

3. How do we get a character at a certain index?

s.charAt(index)

4. How do we get the index of a certain character?

s.indexOf(“char”)

5. How do we trim a string down to a certain length?

s.substring(start index) // this one has end as the last index
s.substring(start index, end index)

6. What does String's comparTo method evaluate?

Length of the string and value of the characters (including upper/lower case difference).

7. StringBuilder vs String: how do their concatenation methods differ?

String concatenation has a new string recreated with each concatenation, which while easier to implement can be costly with large calculations. StringBuilder, however, just adds on to each StringBuilder without creating a new one.

8. Methods of class StringBuilder

Syntax Topics

1. Enhanced for loop vs regular for loop: structure and syntax

Enhanced for loop needs objects.

Say we have ArrayList<File> files, ArrayList<BigInteger> bigInts

```
for (int i = 0; i < files.size(); i++){
	System.out.print(files.get(i).toString());
}

for (File f : files){
	System.out.print(f.toString());
}

String res = "";

for (int i = 0; i < bigInts.size(); i++){
	res = res + " " + bigInts.get(i).toString();
```

```
}

res = "";

for (BigInteger b : bigInts){
        res = res + " " + b.toString();
}
```

2. Inner classes

Inner classes are useful for encapsulation and data hiding, when we want some custom data structure or other functioning inside our class but don't want it visible to any other classes. If we declare our inner class private, nothing else can touch it.

```
public class SomeClass{
        ...

        private class AnotherClass{
        things
        }
}
```

Here, SomeClass can still access all the private instance variables of AnotherClass, but no other class will be able to use it.

Other Topics

1. Recursion basics

When using recursion, you have to stop it some way. In the example of the recursive order traversals, it was stopped since the recursive statement was behind a selection statement that would eventually evaluate to false; another way to stop it would be with a base case, which should eventually be reached by changing the recursive call to meet the base case.

2. Class Comparable

Class Comparable is an interface that provides functionality for comparison. Some classes that implement Comparable are Character (all wrapper classes, actually), String, Date, BigInteger, AccessMode.

3. Methods of class Comparable

compareTo is the only method.

This.compareTo(that) returns an int: a negative number if this is less than that, 0 if this and that are equal, and a positive number if this is greater than that.

A check for comparison this way could be
if (this.compareTo(that) < 0){ do something}
if (this.compareTo(that) > 0){ etc }