COMPUTER SCIENCE DEPARTMENT PICNIC

Welcome to the 2016-2017 Academic year !

Meet your faculty, department staff, and fellow students in a social setting. Food and drink will be provided.

When: Saturday, September 10th
Time: 11am – 2pm
Where: City Park Shelter #7

Operations

- Push the power button and hold.
- Once the light begins blinking, enter the room code
  - This room's code is BC
- When a question is asked, you have 30 seconds to respond
- Enter the letter of the appropriate answer
- When you enter the letter of the answer, your i-clicker will blink green.
- It is your responsibility to check for that green light.

I Forgot…

- If you forgot your IClicker, or your batteries fail during the exam
  - Your worst quiz score is not counted to cover this situation.
  - All other quizzes count.
- If you have an excused absence, you may have the quiz score exempted.

IC Question 1

- Why is abstraction a strength when we program?

A. It allows us to identify where we use classes
B. It allows us to use objects without knowing how they work
C. It allows us to use variables without knowing how they work
D. All of the above

IC Question 1 Answer

- Why is abstraction a strength when we program?

A. It allows us to identify where we use classes
B. It allows us to use objects without knowing how they work
C. It allows us to use variables without knowing how they work
D. All of the above

IC Question 2

- If we have defined a class to provide functionality to client code, what is the purpose of the main method is that class?

A. To provide a mechanism for unit testing
B. To provide print statements
C. To allow the programmer to build the class
D. None of the above

IC Question 2 Answer

- If we have defined a class to provide functionality to client code, what is the purpose of the main method is that class?

A. To provide a mechanism for unit testing
B. To provide print statements
C. To allow the programmer to build the class
D. None of the above

7

IC Question 3

```
public <type> ( <parameter(s)> ) {
    <statement(s)> ;
}
```

- For a constructor the <type> is which of the following:

A. The return type
B. The method type
C. The name of the class
D. The type of the parameter

8

IC Question 3 Answer

```
public <type> ( <parameter(s)> ) {
    <statement(s)> ;
}
```

- For a constructor the <type> is which of the following:
  A. The return type
  B. The method type
  C. The name of the class
  D. The type of the parameter

9

IC Question 4

- Instance variables can be declared *which of the following* to indicate that no code outside their own class can access or change them.
  A. Public
  B. Instance
  C. Class
  D. Private
  E. None of the above

10

IC Question 4 Answer

- Instance variables can be declared *which of the following* to indicate that no code outside their own class can access or change them.

A. Public
B. Instance
C. Class
D. Private
E. None of the above

IC Question 5

- **When you see this used like below, what is occurring?**

```
this(parameters);
```

A. Referring to an instance variable
B. Calling a method
C. Calling a constructor from another constructor
D. Calling a static method

IC Question 5 Answer

- **When you see this used like below, what is occurring?**

```
this(parameters);
```

A. Referring to an instance variable
B. Calling a method
C. Calling a constructor from another constructor
D. Calling a static method

13

ArrayLists

Chapter 12.1 in Savitch

Using arrays to store data

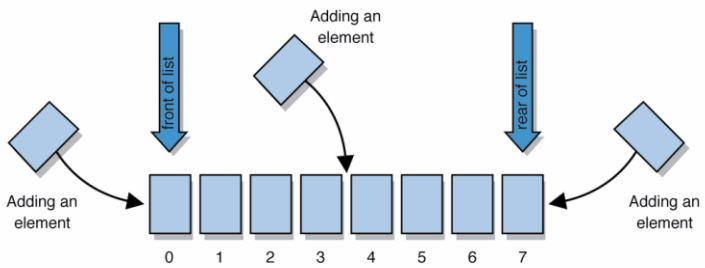
- Arrays: store multiple values of the same type.
- Conveniently refer to items by their index
- Need to know the size before declaring them:

```
int[] numbers = new int[100];
```

- We often need to store an unknown number of values.
  - Need to either count the values or resize as additional storage space is needed.

Lists

- **list**: a collection storing an ordered sequence of elements,
  each accessible by a 0-based index
  - a list has a **size** (number of elements that have been added)
  - elements can be added at any position



ArrayIntList

- Let's consider the methods of a class called `ArrayIntList` that represents a list using `int[]`
  - behavior:
    - `add(`**value**`),` `add(`**index, value**`)`
    - `get(`**index**`),` `set(`**index, value**`)`
    - `size()`
    - `remove(`**index**`)`
    - `indexOf(`**value**`)`

      ...
  - The list's *size* will be the number of elements added to it so far

ArrayIntList

- construction

```
int[] numbers = new int[5];
ArrayIntList list = new ArrayIntList();
```

- storing a given value:

```
numbers[0] = 42;
list.add(42);
```

retrieving a value

```
int val = numbers[0];
int val = list.get(0);
```

- searching for a given value

```
for (int i = 0; i < numbers.length; i++) {
    if (numbers[i] == 27) { ... }
}
if (list.indexOf(27) >= 0) { ... }
```

Pros/cons of `ArrayIntList`

- pro (benefits)
  - simple syntax
  - don't have to keep track of array size and capacity
  - has powerful methods (`indexOf`, `add`, `remove`, `toString`)
- con (drawbacks)
  - `ArrayIntList` only works for `int`s (arrays can be any type)
  - Need to learn how to use the class

Java Collections and ArrayLists

- Java includes a large set of powerful classes that provide functionality for storing and accessing collections of objects
- The most basic, `ArrayList`, can store any type of Object.
- All collections are in the `java.util` package.

```
import java.util.ArrayList;
```

Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- When constructing an `ArrayList`, you can specify the type of elements it will contain between < and >.
  - We say that the `ArrayList` class accepts a *type parameter*, or that it is a *generic* class.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Alice");
names.add("Bob");
```

`ArrayList` methods

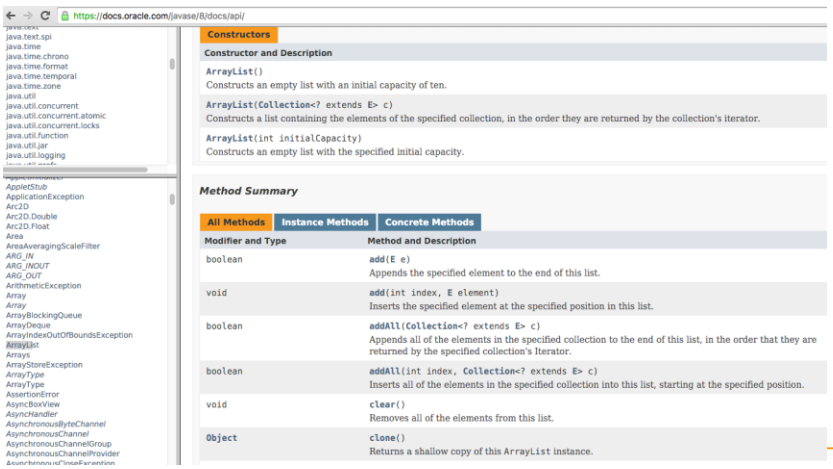
`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value at given index, shifting subsequent values right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

`ArrayList` methods 2

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from the given list at the end of this list inserts the list at the given index of this list
`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from**`, `**to**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns an array of the elements in this list

Learning about classes

- The Java API specification website contains detailed documentation of every Java class and its methods.



https://docs.oracle.com/javase/8/docs/api/

Iterating through an array list

- Suppose we want to look for a `value` in an ArrayList of Strings.

```
for (int i = 0; i < list.size(); i++) {
    if(value.equals(list.get(i)){
        //do something
    }
}
```

- Alternative:

```
for (String s : list) {
    if(value.equals(s)){
        //do something
    }
}
```

Note - generics in Java 7+ and above

In version 7+ of Java, rather than doing:

```
ArrayList<Type> name = new ArrayList<Type>();
```

You can save a few keystrokes:

```
ArrayList<Type> name = new ArrayList<>();
```

Modifying while looping

- Consider the following flawed pseudocode for removing elements that end with 's' from a list:

```
removeEndS(list) {
    for (int i = 0; i < list.size(); i++) {
        get element i;
        if it ends with an 's', remove it.
    }
}
```

  - What does the algorithm do wrong?

index	0	1	2	3	4	5
value	"she"	"sells"	"seashells"	"by"	"the"	"seashore"
size	6					

ArrayList of primitives?

- The type you specify when creating an `ArrayList` must be an object type; it cannot be a primitive type.
  - The following is illegal:

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

- But we can still use `ArrayList` with primitive types by using special classes called *wrapper* classes in their place.

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

Wrapper classes: Example

- Every java primitive has a class dedicated to it.

```
Example:

int x = 3;
Integer y = new Integer(5);

int z = x + y;

int z  = x + y.intValue();  // convert wrapper to primitive

// can also construct an Integer from a string:

y = new Integer("5");
```

29

ArrayLists of wrapper type objects

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean
float	Float

- A wrapper is an object whose purpose is to hold a primitive value and to provide more functionality.
- Once you construct the list, use it with primitives as normal (autoboxing):

```
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
```

ArrayLists of wrapper type objects

- Autoboxing:

```
ArrayList<Double> grades = new ArrayList<Double>();
// Autoboxing: create Double from double 3.2
grades.add(3.2);
grades.add(2.7);
double sum = 0.0;
for (int i = 0; i < grades.size(); i++) {
   //AutoUNboxing from Double to double
   sum += grades.get(i);
}
...
```

Java Collections

- ArrayList belongs to Java's Collections framework.
- Other classes have a very similar interface, so it will be easier to learn how to use those classes once you've learned ArrayList

IC Question 6

- Java includes a large set of powerful classes that provide functionality for storing and accessing collections of which of the following?

A. Classes
B. Objects
C. Variables
D. Methods

IC Question 6 Answer

- Java includes a large set of powerful classes that provide functionality for storing and accessing collections of which of the following?

A. Classes
B. Objects
C. Variables
D. Methods

IC Question 7

- To specify an ArrayList of a primitive variable type, what must you use?

A. Static variables
B. Static methods
C. Public methods
D. Wrapper classes

IC Question 7 Answer

- To specify an ArrayList of a primitive variable type, what must you use?

A. Static variables
B. Static methods
C. Public methods
D. Wrapper classes

IC Question 8

- We say that the `ArrayList` class accepts a *type parameter*, or that it is a _____ class.

A. Wrapper
B. Generic
C. Static
D. Public

IC Question 8 Answer

- We say that the `ArrayList` class accepts a *type parameter*, or that it is a _____ class.

A. Wrapper
B. Generic
C. Static
D. Public

Looking ahead: Interfaces

- A Java interface specifies which public methods are available to a user
- A class implements an interface if it provides all the methods in the interface
- Interfaces allow for common behavior amongst classes. Example: the List interface is implemented by several Collections classes (LinkedList, ArrayList, Vector, Stack)