This recitation

- An interesting point about A3: Using previous methods to avoid work in programming and debugging. How much time did you spend writing and debugging prepend?
- Enums (enumerations)
- Generics and Java's Collection interfaces and classes

How to use previous methods in A2

The A2 handout contained this:

**Further guidelines and instructions!**

"Note that some methods that you have to write …. Also, in writing methods 4..7, writing them in terms of calls on previously written methods may save you time."

Did you read that? Think about it? Attempt it?

A lesson in:
1. Reading carefully, wisely.
2. Thinking about what methods do, visualizing what they do.

About enums (enumerations)

An enum: a class that lets you create mnemonic names for entities instead of having to use constants like 1, 2, 3, 4

The declaration below declares a class **Suit**.
After that, in any method, use **Suit.Clubs**, **Suit.Diamonds**, etc. as constants.

```
public enum Suit {Clubs, Diamonds, Hearts, Spades}
```

Testing for an enum constant

```
public enum Suit {Clubs, Diamonds, Hearts, Spades}
Suit s = Suit.Clubs;
Then
s == Suit.Clubs is true
s == Suit.Hearts is false
```

```
switch(s) {
    case Clubs:
        color = "black"; break;
    case Diamonds:
        color = "red"; break;
    case Hearts:
        color = "red"; break;
    }
```

Miscellaneous points about enums

```
public enum Suit {Clubs, Diamonds, Hearts, Spades}
```

This declaration is shorthand for a class that has a constructor, four constants (public static final variables), a static method, and some other components. Here are some points:

1. **Suit** is a subclass of **Enum** (in package java.lang)
2. It is not possible to create instances of class **Suit**, because its constructor is private!
3. It’s as if **Clubs** (as well as the other three names) is declared within class **Suit** as

```
public static final Suit Clubs = new Suit(some values);
```

You don’t care what values
Miscellaneous points about enums

```java
public enum Suit {Clubs, Diamonds, Hearts, Spades}
```

4. Static function `values()` returns a `Suit[]` containing the four constants. You can, for example, use it to print all of them:

```java
for (Suit s : Suit.values())
    System.out.println(s);
```

Output:

```
Clubs
Diamonds
Hearts
Spades
```

Can save this array in a static variable and use it over and over:

```java
private static Suit[] mine = Suit.values();
```

5. Static function `valueOf(String name)` returns the enum constant with that name:

```java
Suit c = Suit.valueOf("Hearts");
```

After the assignment, `c` contains (the name of) object Hearts

```
Suit@2
```

This is the object for Hearts:

```
Suit@2
```

6. Object Clubs (and the other three) has a function `ordinal()` that returns its position in the list

```java
Suit.Clubs.ordinal() is 0
Suit.Diamonds.ordinal() is 1
```

We have only touched the surface of enums. E.g. in an enum declaration, you can write a private constructor, and instead of `Clubs` you can put a more elaborate structure. All this is outside the scope of CS2110.

---

Package `java.util` has a bunch of classes called the Collection Classes that make it easy to maintain sets of values, list of values, queues, and so on. You should spend some time looking at their API specifications and getting familiar with them.

Remember:

- A set is a bunch of distinct (different) values. No ordering is implied
- A list is an ordered bunch of values. It may have duplicates.

---

**Interface Collection:** abstract methods for dealing with a group of objects (e.g. sets, lists)

**Abstract class AbstractCollection:** overrides some abstract methods with methods to make it easier to fully implement Collection

**AbstractList, AbstractQueue, AbstractSet, AbstractDeque** overrides some abstract methods of AbstractCollection with real methods to make it easier to fully implement lists, queues, set, and deques

Next slide contains classes that you should become familiar with and use. Spend time looking at their specifications. There are also other useful Collection classes.
ArrayList v = new ArrayList();

An object of class ArrayList contains a growable/shrinkable list of elements (of class Object). You can get the size of the list, add an object at the end, remove the last element, get element i, etc. 

Fields that contain a list of objects (o_0, o_1, ..., o.size()-1)
ArrayList () add(Object) get(int) size() remove(…) set(int, Object) ...

You can get the size of the list, add an object at the end, remove the last element, get element i, etc. More methods exist! Look at them!

HashSet s = new HashSet();

An object of class HashSet contains a growable/shrinkable set of elements (of class Object). You can get the size of the set, add an object to the set, remove an object, etc. More methods exist! Look at them!

Fields that contain a set of objects {o_0, o_1, ..., o.size()-1}
HashSet() add(Object) contains(Object) size() remove(Object) ...

Don’t ask what “hash” means. Just know that a Hash Set object maintains a set.

Iterating over a HashSet or ArrayList

HetSet s = new HashSet();

… code to store values in the set …

for (Object e : s) {
    System.out.println(e);
}

A loop whose body is executed once with e being each element of the set. Don’t know order in which set elements processed.

Use same sort of loop to process elements of an ArrayList in the order in which they are in the ArrayList.

Interface List: abstract methods for dealing with a list of objects (o_0, …, o_n-1). Examples: ArrayList, LinkedList

Abstract class AbstractList: overrides some abstract methods with real methods to make it easier to fully implement List

Homework: Look at API specifications and build diagram giving format of HashSet

Generics and Java’s Collection Classes

generic adjective ˈjənərık, -rık\  relating or applied to or descriptive of all members of a genus, species, class, or group: common to or characteristic of a whole group or class; typifying or subsuming: not specific or individual.

From Wikipedia: generic programming: a style of computer programming in which algorithms are written in terms of to-be-specified-later types that are then instantiated when needed for specific types provided as parameters.

In Java: Without generics, every ArrayList object contains a list of elements of class Object. Clumsy

With generics, we can have an ArrayList of Strings, an ArrayList of Integers, an ArrayList of Genes. Simplifies programming, guards against some errors.
**Generics: say we want an ArrayList of only one class**

API specs: ArrayList declared like this:

```java
public class ArrayList<E> extends AbstractList<E>
    implements List<E> … { … }
```

Means:
Can create ArrayList specialized to certain class of objects:

```java
ArrayList<String> vs= new ArrayList<String>(); //only Strings
ArrayList<Integer> vi= new ArrayList<Integer>(); //only Integers
```

These are illegal:
```java
vs.add(3);   //vs.get(0) has type String
vi.add("abc");  //No need to cast
```

**Generics allow us to say we want ArrayList of Strings only**

API specs: ArrayList declared like this:

```java
public class ArrayList<E> extends AbstractList<E>
    implements List<E> … { … }
```

Full understanding of generics is not given in this recitation.
E.g. We do not show you how to write a generic class.

**Important point:** When you want to use a class that is defined like ArrayList above, you can write

```java
ArrayList<C> v= new ArrayList<C>(…);
```

to have v contain an ArrayList object whose elements HAVE to be of class C, and when retrieving an element from v, its class is C.

**ArrayList to maintain list of Strings is cumbersome**

ArrayList v= new ArrayList();

... Store a bunch of Strings in v ...

String ob= ((String) v.get(0)).length();

All elements of v are of type Object.
So, to get the size of element 0, you first have to cast it to String.

Make mistake, put an Integer in v?
May not catch error for some time.

v ArrayList@x1 ArrayList

--- Only Strings, nothing else

ArrayList @x1 Object

Fields that contain a list of objects (o0, o1, …, osize()-1)
ArrayList() add(Object) get(int) size()
remove() set(int, Object) …