Lecture 6: Packages, Java API
The story so far

- Imperative Programming
  - Statements - Lecture 1
  - Methods - Lecture 2
  - Arrays - Lecture 2
- Object Oriented Programming
  - Objects, fields and methods - Lecture 3
  - Access modifiers: public and private - Lecture 3
  - Class Inheritance - Lecture 4
  - Static, interface inheritance - Lecture 5
Questions from last class

- What’s the point of interfaces?
- Why would someone ever use static?
Assignment 5

- Set should not inherit from Bag.
- Code reuse: The add() and remove() methods in Set are good places to use Set.contains().
- Make sure to follow the description of each method in an interface exactly. Interfaces will often have lengthy descriptions about exactly what the method is supposed to do, what corner cases it is supposed to handle, etc.
Today

Writing real useful code.
Overview

- Exception Handling
- Generics
- Packages
- API’s
Exception Handling

- Things always go wrong:
  - Accessing beyond the length of the array
  - Accessing a field of an object that does not hold anything (null object)
- How do you know when things go wrong?
  - The dreaded Null pointer exception
  - Turns out this is a specific case of a more general error handling mechanism
Exception Handling

- **Expressive** way of handling errors that allow you to either:
  - Provide enough information to fix the error soon
  - Recover and move on if things do go wrong.
Exception Handling Syntax

```java
try {
    int var=a[10];
}

catch(Exception e) {
    System.out.println(e.info());
}
```

Type of Exception

Any meaningful code
The Exception hierarchy

- Exception
  - ClassNotFoundException
  - NullPointerException
  - RuntimeException
    - IndexOutOfBoundsException
      - ArrayIndexOutOfBoundsException
Throwing exceptions

- You can create your own exceptions by deriving from one of these classes in the tree.
- Create an exception object just like any other object. Throw it for someone else who is using your code to catch:

```java
public checkPositive() throws Exception {
    if (positiveNumber < 0) {
        throw new Exception();
    }
}
```
Overview

- Exception Handling
- Generics
- Packages
- API’s
Generics

- Consider the Bag and Set examples from last assignment
- Bag/Set contained elements that were int
- What if we want the Bag/Set to contain:
  - boolean
  - double
  - String
  - Runners
  - Bank Accounts
The obvious solution

- Write new code for
  - BagInt
  - Bagdouble
  - Bagboolean
  and the corresponding collections they implement
- But what is new about each of these code pieces?
  - In all likelihood, you are going to copy, paste and modify
  - The machine should be doing this automatically for me!!
Generics

- Write code for several classes all at once:

```java
class Bag<E> extends Collection<E> {
    E[] storageArray;
}

public interface Collection<E> {
    void add(E element);
    void remove(E element);
    boolean contains(E element);
    boolean isInvalid(E element);
}
```

Type parameter
Generics: Internals

- All instantiations of a generic class share the same copy of code.
- This code operates on *Objects*, irrespective of the instantiation of the generic class.
- *Object* is a class that is right at the top of the inheritance tree.
- Every non-primitive/class inherits from the class *Object*
  - By the subset-superset analogy, an instance or object of any class can be used where an object of type *Object* is required.
Generics : Caveat

- Constructor :

  Correct :

  ```java
  public Bag(E inputArray) {
      storageArray = inputArray;
  }
  ```

  Wrong :

  ```java
  public Bag(int size) {
      storageArray = new E[size];
  }
  ```

  There is only one copy of the class at runtime for all instantiations. That copy doesn’t know which E you are referring to, since it could be any class and the code replaces E with Object.
Overview

- Exception Handling
- Generics
- Packages
- API’s
Motivating packages

- A group of related statements makes a function.
- A group of related functions makes a class.
- A group of related classes is a package.
Packages

- Each class belongs to a package
- Classes in the same package serve a similar purpose
- Physically:
  - Packages are just directories
  - Classes in other packages need to be imported
Defining packages

package path.to.package.foo;

class Foo {

    ...

}

Using Packages

```python
import path.to.package.foo.Foo;
import path.to.package.foo.*;
```
package parenttools;

public class BabyFood {
}

package parenttools;

public class Baby {
}
package adult;

import parenttools.Baby;
import parenttools.BabyFood;

public class Parent {
    public static void main(String[] args) {
        Baby baby = new Baby();
        baby.feed(new BabyFood());
    }
}

Why Packages?

- Combine similar functionality
  - org.boston.libraries.Library
  - org.boston.libraries.Book

- Fewer lines of code:
  - Import org.* (all classes in org)

- Create a new namespace & allows you to isolate the same names to within a package:
  - shopping.List
  - packing.List
Special Packages

All classes “see” classes in the same package
(no import needed)

All classes “see” classes in java.lang

Example: java.lang.String; java.lang.System
Recall System.out.println()
Aside: Example of static

- System.out.println:
  - System is a class belonging to java.lang
  - Out is a **static** field of type PrintStream
  - println is a member function of PrintStream
  - The console is shared between all classes in the program so it makes sense to keep it’s representations (out and in) **static**
Overview

- Exception Handling
- Generics
- Packages
- API’s
Application Programming Interface

- We have been talking a lot about code reuse.
- In its most general form, an API is a way to get software components to work together.
- Reuses code to avoid extra debugging
- Reduces development time for new apps
An example: The Java API

- [http://java.sun.com/javase/6/docs/api/](http://java.sun.com/javase/6/docs/api/)
- In Java’s case, it’s a group of packages
- Provides operations such as:
  - File I/O
  - Graphics
  - Collections (List, Map, Set)
  - Cryptography
  - User Interfaces
Motivating the Java API

What if you need an array but don’t know it’s size to begin with?

The obvious solution:

Create the array bigger than you need

Track the next “available” slot

Book[] books = new Book[10];

int nextIndex = 0;

books[nextIndex] = b;

nextIndex = nextIndex + 1;

What if the library expands?
ArrayList

Modifiable list

Internally implemented with arrays

- Get/put items by index
- Add items
- Delete items
- Loop over all items
- Think of it like Bag from the previous assignment.
Array → ArrayList

```java
Book[] books = new Book[10];
int nextIndex = 0;
books[nextIndex] = b;
nextIndex += 1;

ArrayList<Book> books = new ArrayList<Book>();
books.add(b);
```
import java.util.ArrayList;

class ArrayListExample {

    public static void main(String[] arguments) {
        ArrayList<String> strings = new ArrayList<String>();
        strings.add("Evan");
        strings.add("Eugene");
        System.out.println(strings.size());
        System.out.println(strings.get(0));
        System.out.println(strings.get(1));
        strings.set(0, "Goodbye");
        strings.remove(1);
        for (int i = 0; i < strings.size(); i++) {
            System.out.println(strings.get(i));
        }
    }
}
Q2: Why are interfaces useful?

- Consider a sorting algorithm:

```java
for (i=1; i <= length(A)-1; i++) {
    key = A[i];
    j = i - 1;
    while (j >= 0 && A[j] > key) {
        j = j - 1;
    }
    A[j + 1] = key;
}
```
for (i=1; i <= length(A)-1; i++) {
    key = A[i];
    j = i - 1;
    while (j >= 0 && A[j] > key) {
        j = j - 1;
    }
    A[j+1] = key;
}

• Observe:
  • It doesn’t matter what A[j] stores floats, ints, etc or even some user-defined object
  • All that matters is that there is some sensible way of comparing A[j] and key.
  • So long as this exists, sorting can proceed independent of what A stores
Java API: Comparable\(<E>\>

- An interface that specifies exactly one method: `compareTo` which tells the Java API how to compare two elements of type `E`.
- Many other interfaces/classes use `Comparable\(<E>\>` as a black box:
  - Can sort arbitrary arrays now.
- Can define this for user defined types:
  - Runners
  - Tuples
The Interface SortedSet\<E\>
import java.util.TreeSet;

class SetExample {
    public static void main(String[] arguments) {
        TreeSet<String> strings = new TreeSet<String>();
        strings.add("Evan");
        strings.add("Eugene");
        strings.add("Adam");

        System.out.println(strings.size());
        System.out.println(strings.first());
        System.out.println(strings.last());

        strings.remove("Eugene");

        for (String s : strings) {
            System.out.println(s);
        }
    }
}
Thank you 😊

- Assignment 6 is up. Due at 3 pm tomorrow.
- All incomplete assignments due by midnight on Monday (ie 23:59 on Jan 30th)
- Feedback on the course is most welcome 😊
- HKN evaluation link to be sent over email.