CS 10: Problem solving via Object Oriented Programming

Winter 2017

Tim Pierson
260 (255) Sudikoff

Day 6 – Lists

Agenda

1. Defining an ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Abstract Data Types specify operations on a data set that defines overall behavior

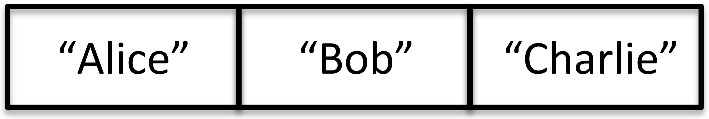
Abstract Data Types (ADTs)

- ADTs specify a set of operations (insert, remove, etc) that define how the ADT behaves on a collection of data
- At the ADT level we don't know (and don't really care) how data elements are stored (e.g., linked list or array, doesn't matter) or what kind of data they hold (e.g., Strings, integers, objects). This is the *Abstract* in Abstract Data Type
- Idea is to hide the way the data are represented while allowing others to work with the data in a predictable manner

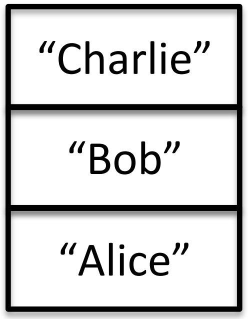
The same operation can act differently in different ADTs, defining unique behavior

Examples of List, Stack, and Queue ADTs

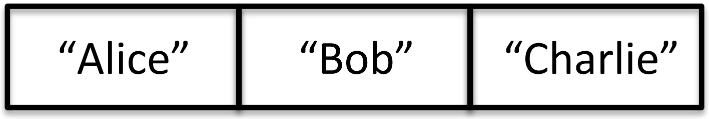
List



Stack



Queue



Behavior

- ***Insert*** anywhere
- ***Remove*** from anywhere
- Keeps elements in order

- ***Insert*** only at top
- ***Remove*** only from top
- "LIFO"

- ***Insert*** only at end
- ***Remove*** only from front
- "FIFO"

An Interface defines the set of operations required to implement an ADT

Interface

- Defines a set of operations that MUST be implemented (if you're going to be an ADT of a particular type, you'll have to implement these functions)

- Does not specify HOW to implement the functionality (use an array, use a linked list – its all up to you, Interface doesn't care)

- Cannot "*new*" an Interface -- it has not implementation!

- Today we focus on the List ADT implemented as linked list, soon will cover other ADTs such as stacks, queues, trees, and graphs.

- Tomorrow we will look at an array implementation

The List Interface describes several operations, but not implementations

List ADT

Operation	**Description**
`size()`	Return number of items in List
`isEmpty()`	True if no items in List, otherwise false
`get(i)`	Return the item at index *i*
`set(i,e)`	Replace the item at index *i* with item *e*
`add(i,e)`	Insert item *e* at index *i*, moving all subsequent items one index later
`remove(i)`	Remove and return item at index *i*, move all subsequent items one index earlier

These operations MUST be implemented to complete the ADT
Free to implement other methods, but must have these
Notice the familiar look from Java's ArrayList

Interfaces go in one file, implementations go in another file

Interface file
Specifies required operations
`SimpleList.java`

Uses keyword
`interface`

Linked list implementation
`SinglyLinked.java`

OR

Array implementation

Implementation file
Actually implements required operations using a specific data structure

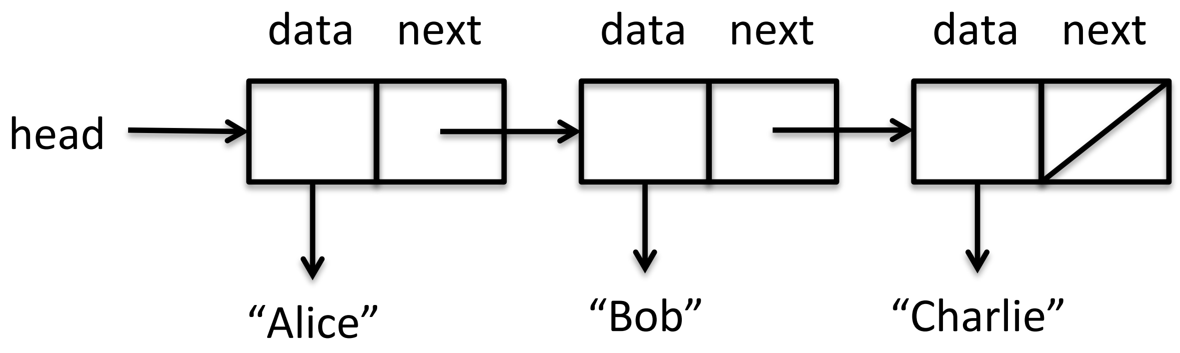
Same interface *could* be implemented in different ways (e.g., linked list *or* array)

Class uses keyword
`implements`

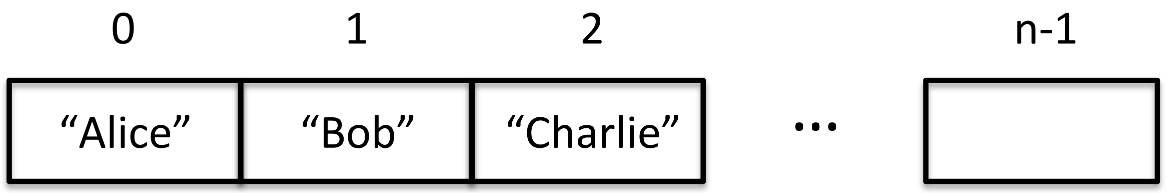
The List ADT could be *implemented* with a singly linked list or an array; either works

Examples of List implementation

Singly linked list



Array



Agenda

1. Defining an ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Generics allow a variable to stand in for a Java type

Interface declaration

```
public interface SimpleList<T> {
   ...
   public T get(int index) throws Exception;
   public void add(int index, T item) throws Exception;
}
```

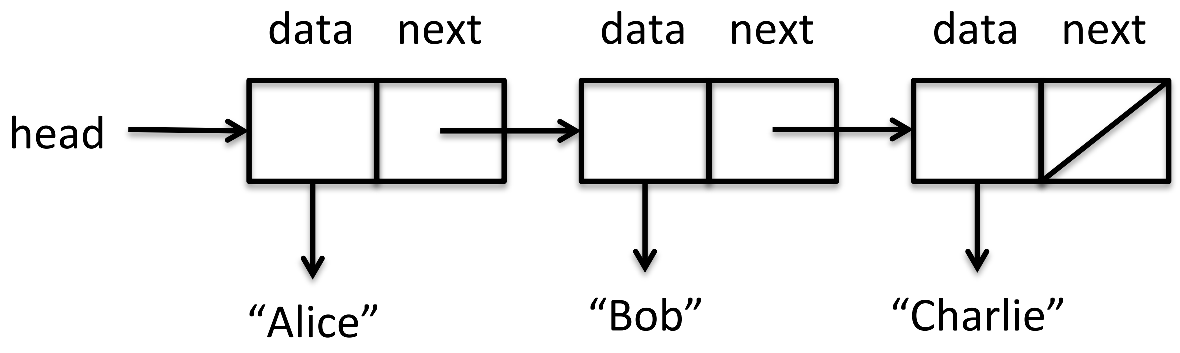
- T stands for whatever object *type* we instantiate
- SimpleList<Blob> then T always stands for Blob
- SimpleList<Point> then T always stands for Point
- Allows us to write one implementation that works regardless of what kind of object we store in our data set
- Must use class version of primitives (Integer, Double, etc)
- Typically name type variables with a single uppercase letter, often T for "type", but sometimes E for "element", or as we'll see later K and V for "key" and "value", and V and E for "vertex" and "edge”

Agenda

1. Defining an ADT
2. Generics
3. ➡ Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

Singly linked list review: elements have data and a next pointer

Singly linked list

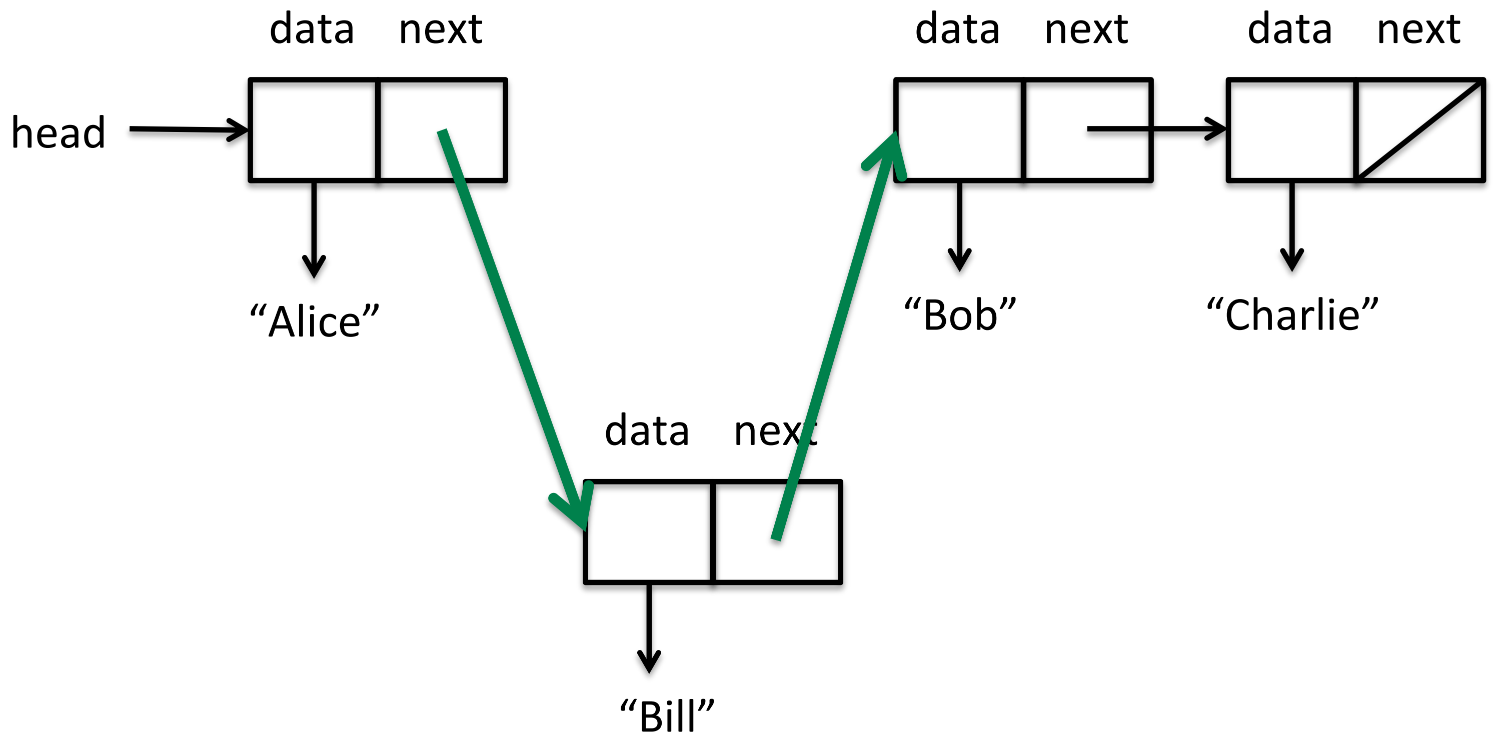


Finding data in Singly Linked List

- Keep pointer to head
- To find item, must start at head and march down until get to desired index (or in other implementations find object with matching data – find “Charlie” vs. get at index 2)

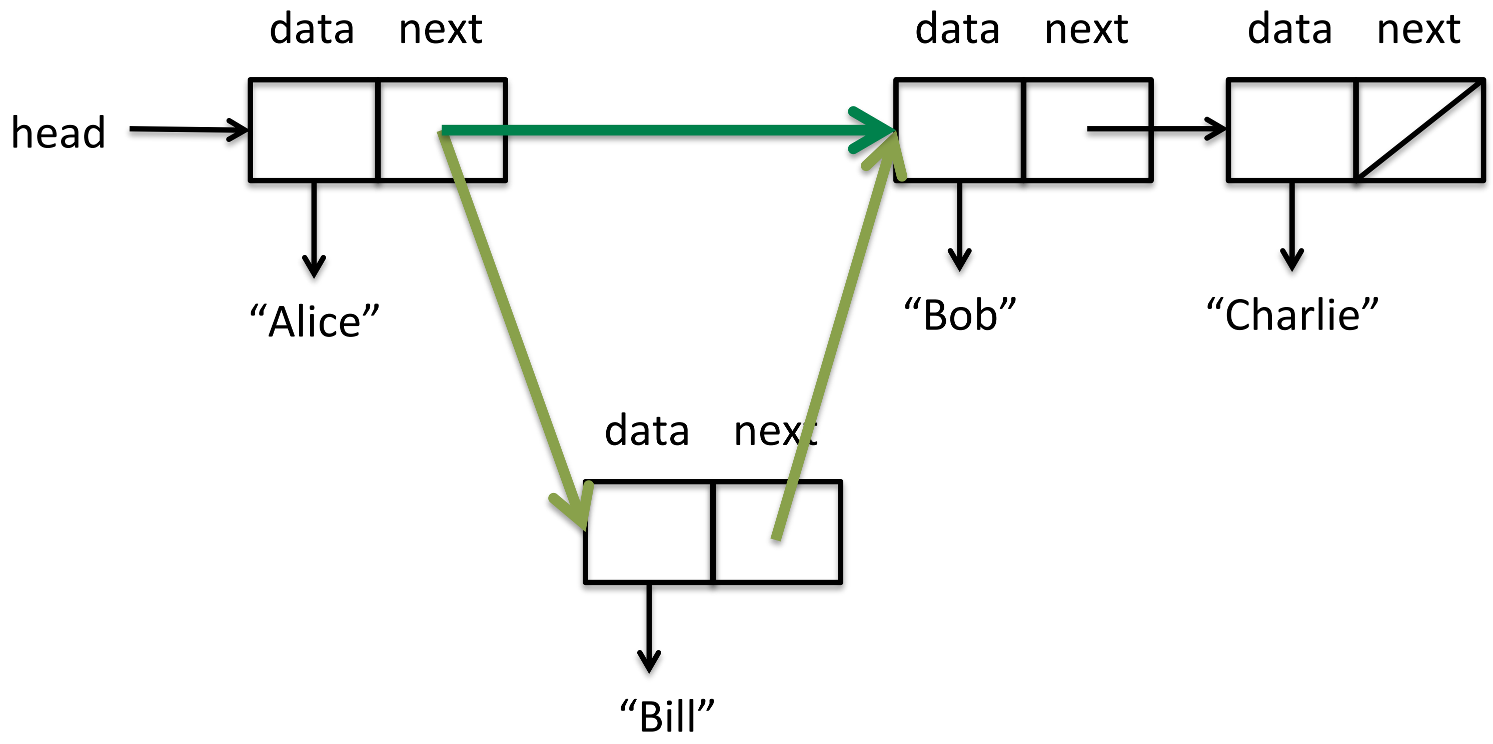
Insert "splices in" a new object anywhere in the list by updating two pointers

Insert item at index 1



Remove takes an item out of the list by updating one pointer

Remove item at index 1



SimpleList.java defines a List Interface SinglyLinked.java implements as linked list

SimpleList.java defines Interface

- *`size()`* *return* number of elements stored in the list
- *`add(int idx, T item)`* adds *item* at index position *idx*
- *`remove(int idx)`* removes item at index position *idx*
- *`get(int idx)`* return the item at index *idx*
- *`set(int idx, T item)`* replace item at index *idx* with item

SinglyLinked.java implements Interface as a linked list

- Implements SimpleList Interface as a singly linked list, so must implement all methods in Interface; can add more methods
- Defines a nested class for elements in list
- Each element has a `data` instance variable of type T and a `next` pointer
- Keeps a pointer to `head,` uses `advance(int n)` to get to item *n*
- `add()`,`remove()` use `advance()` to find previous item
- `toString()` for println

ListTest.java uses implementation to keep track of items

ListTest.java

- Create `new SinglyLinked` to hold `Strings`, so T stands for String in SinglyLinked
- Add items (Strings)
- Print list (remember: println calls `toString()`, implemented in SinglyLinked.java)
- Run

Agenda

1. Defining an ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

An exception indicates that something unexpected happened at run-time

Cannot check for all errors at compile time

What if we ask for element at index -1 of an array?

- There is no clear, “always-do-this”, answer
- Maybe we should return null
- Maybe we should stop program execution

Exceptions provide a way to show something is amiss, and let calling functions deal with error (or not)

“Throw” error with `throw new Exception(“error description”)`

Java provides structured error-handling via try/catch blocks

- Catch block specifies type of error it handles
- Catch executes only if error in try body
- Can have multiple catch blocks for each try
- Finally block executes regardless whether try succeeds or fails
- Exceptions not handled before *`main()`* kill execution

Exceptions can be handled at run time with try/catch/finally blocks

ListExceptions.java

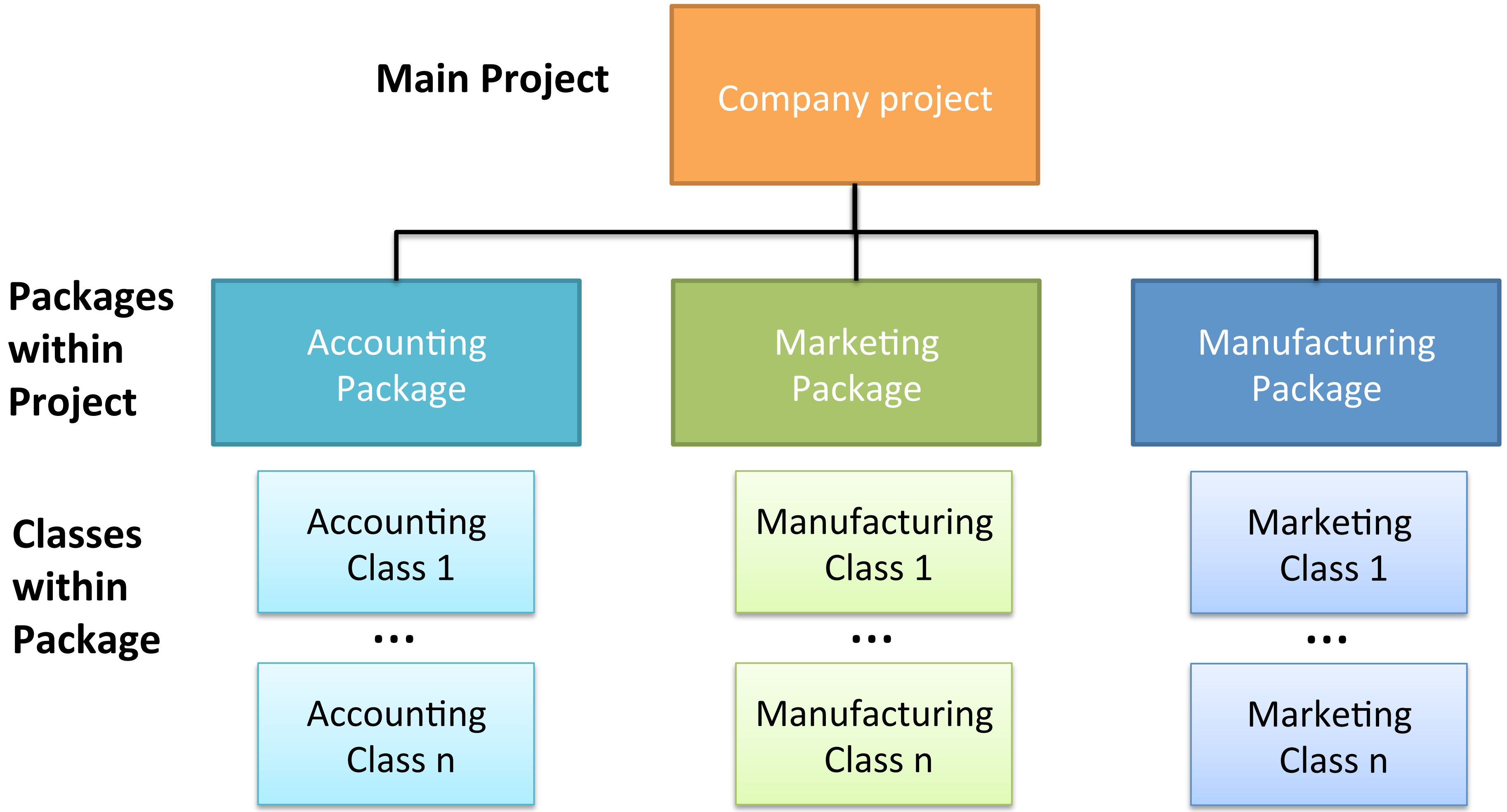
- Create new SinglyLinked
- Add items to list
- Before remove calls, list contains z->a->b->e->[/]
- NOTE the set at line 13, not an add!
- After removes list contains a->[/]
- Cause errors and see catch in action
- Finally always called
- Exceptions thrown by SinglyLinkedList.java (e.g., line 49)
- If method throws exception, must by in try/catch block from caller (see line 49 in SinglyLinked.java and any add in ListExceptions.java)
    - Try adding list.add(1,”f”) on line 24 (outside try/catch)

Agenda

1. Defining an ADT
2. Generics
3. Singly linked list implementation
4. Exceptions
5. Visibility: public vs. private vs. protected vs. package

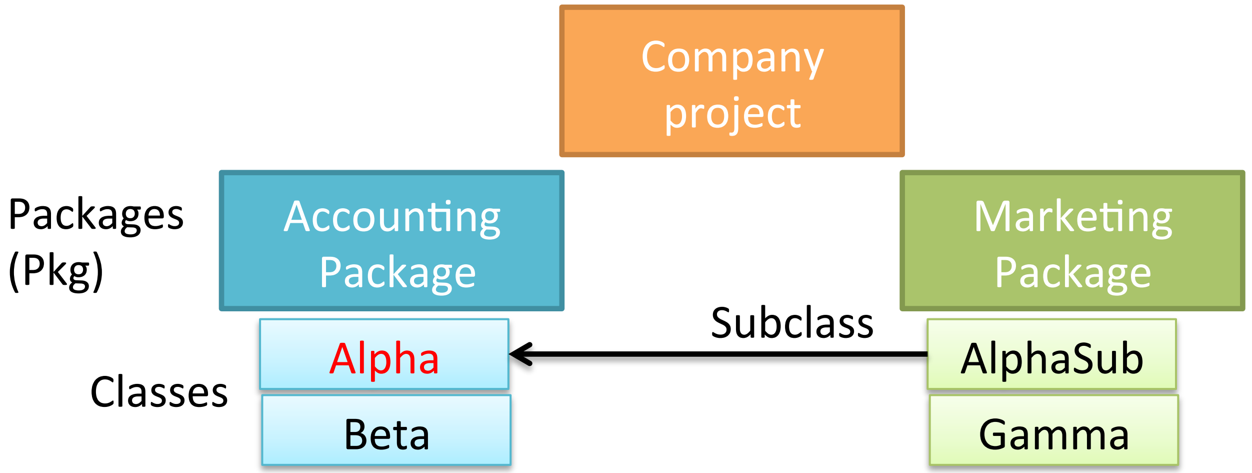
Java allows us to break up major portions of code into Projects, Packages and Classes

Example of master project for a company



Visibility depends on modifier applied

Example: Visibility of Alpha class

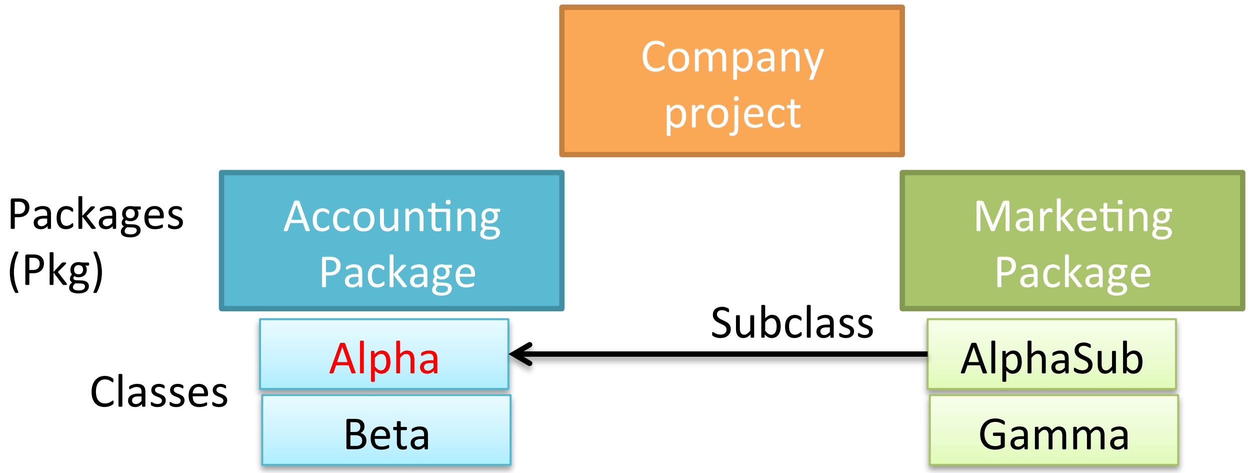


Y = can access
N = cannot access

		Accounting Pkg		Marketing Pkg	
If Alpha is:	**Access by:**	**Alpha**	**Beta**	**AlphaSub**	**Gamma**
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	N	N

Visibility depends on modifier applied

Example: Visibility of Alpha class



Y = can access
N = cannot access

If Alpha is:	Access by:	Accounting Pkg		Marketing Pkg	
		Alpha	**Beta**	**AlphaSub**	**Gamma**
public	Any class	Y	Y	Y	Y
protected	Pkg + Subclass	Y	Y	Y	N
No modifier	Pkg - Subclass	Y	Y	N	N
private	This class only	Y	N	**N**	N