Generic Types and Static Methods

Recall the printList method of ListStuff. Why does it use ? for the generic?

The ? is a wild card. It means that the generic can be anything, and our code does not care what it is.

public static void printList(Iterable<?> list) {

for (Object x : list)

System.out.print(x + " ");

System.out.println();

}

Note that we never really use the generic. We do get Object's out of the list, but everything is an object.

Another example:

Suppose we want to create the following static method (inside the Linked List class):

/\*\* Reverses the nodes of the input list \*/

public static void reverseList(LinkedList<what goes here?> list) {

How to reverse the nodes: by moving the arrows. For a visual image, imagine dealing a deck of cards. As you pull the card off the top of the deck and place it in a pile in front of you,

that card becomes the top of the pile, but the next card you deal goes on top of that. When you are done, you have reversed the deck of cards.

We will do the same by creating a new list "head" node and repeatedly moving the first node of the original list to become the new "head" node of the new list.

To get the arrows right, we really need to draw pictures. Otherwise, we risk losing an arrow or having an arrow point to the wrong thing.

public static void reverse(LinkedList<...> list) {

LLNode<...> newhead = null;

LLNode<...> nodeptr = list.getFront();

while (nodeptr != null) {

LLNode<...> nextptr = nodeptr.getNext(); // so we don't lose the next pointer

nodeptr.setNext(newhead);

newhead = nodeptr;

nodeptr = nextptr;

}

list.setFront(newhead);

}

Can we use ? in the reverse example above? Do we really not care what the generic type is?

No! It turns out that we do care! And it is not obvious.

The lines:

nodeptr.setNext(newhead);

and

list.setFront(newhead);

both care about the generic type. For example, remember the setNext method of LLNode:

public void setNext(LLNode<T> next) {

this.next = next;

}

This code uses the generic to verify that whatever node this node points to stores the same type as this node. Thus, the compiler will not let us use the ? wildcard in reverse

because the compiler needs to typecheck to make sure the generic type of newhead (the node we pass into setNext) matches the generic type of nodeptr.

Likewise, the compiler needs to typecheck to verify the generic on newhead (the node we pass into setFront) matches the generic type of list.

Can we use <Object> for the generic?

No! See the previous lecture, LinkedList<String> cannot be typecast to LinkedList<Object>. The generic specification MUST match so that the compiler can verify that you will not be storing, in this case, non-Strings in a LinkedList<String>.

Inner classes, part 1

An inner class is a class defined inside another class.

A class inside a class is a member of the containing class. Just like methods and fields, it can be public or private (or package or protected), and static or non-static.

Static Inner Classes

Please see the class example List1 for an example of our linked list implementation, but with the linked list node and iterator as static inner classes.

The inner class is public to make it easy to demonstrate how inner classes work, but normally we would make it private because we do not want someone

using the linked list class to be able to access the individual nodes.

Note that, becaue the inner class is static, we need to define a generic for the inner class. We could have used T, but I chose to instead use E for the Node class in order to be

clear that the inner class and outer class generics are different. Why do we need to define another generic? Because the inner class is static. The outer

class generic is only specified on instances of the outer class. The inner class belongs to the entire class, not to a specific instance of the outer class.

Because the inner class is static, an instance of the inner class belongs to the outer class, not an instance of the outer class. So, "this" refers to the

inner class instance.

To create a member of the inner class Node from outside the List1 class, we can use exactly the same Java terminology we use to create other objects and to

access static fields and methods:

List1.Node<String> node = new List1.Node<String>("Hi", null);

Note that the generic goes on the Node class. We do not need to specify the generic for List1 because we are not creating an instance of List1.

(It is not an error to specify the generic for List1, but Java will ignore that generic since no instance is being created.)