Programming Languages and Techniques (CIS120)

Lecture 27
March 23rd, 2016

Generics and Collections
Chapter 25
Announcements

• HW #6 due Tuesday

• I will be away all next week (FP workshop in Germany)
  – Monday's lecture: Yaron Minsky, Jane Street
  – My office hours are cancelled on Monday
  – Guest lecturers Wednesday & Friday, bring clickers!
Java Generics
Subtype Polymorphism*

• Main idea:

Anywhere an object of type A is needed, an object that is a subtype of A can be provided.

```java
// in class C
public static void times2(Counter c) {
    c.incBy(c.get());
}

// somewhere else, Decr extends Counter
C.times2(new Decr(3));
```

• If B is a subtype of A, it provides all of A’s (public) methods.

*polyomorphism = many shapes*
Is subtyping good enough?

Subtype Polymorphism

vs.

Parametric Polymorphism
Mutable Queue ML Interface

```ocaml
module type QUEUE =
 sig
   (* type of the data structure *)
   type 'a queue

   (* Make a new, empty queue *)
   val create : unit -> 'a queue

   (* Add a value to the end of the queue *)
   val enq : 'a -> 'a queue -> unit

   (* Remove the front value and return it (if any) *)
   val deq : 'a queue -> 'a

   (* Determine if the queue is empty *)
   val is_empty : 'a queue -> bool

end
```

How can we translate this interface to Java?
Java Interface

module type QUEUE =
  sig
    type 'a queue

    val create : unit -> 'a queue

    val enq : 'a -> 'a queue -> unit

    val deq : 'a queue -> 'a

    val is_empty : 'a queue -> bool
  end

interface ObjQueue {

    // no constructors
    // in an interface

    public void enq(Object elt);

    public Object deq();

    public boolean isEmpty();
}

interface ObjQueue {
    public void enq(Object elt);
    public Object deq();
    public boolean isEmpty();
}

ObjQueue q = ...;
q.enq(" CIS 120 ");
__A__ x = q.deq();

What type for A?
1. String
2. Object
3. ObjQueue
4. None of the above
Subtype Polymorphism

interface ObjQueue {
    public void enq(Object elt);
    public Object deq();
    public boolean isEmpty();
}

// Is this valid?
No!

// What type for B?
Object

ObjQueue q = ...

q.enq(" CIS 120 ");
Object x = q.deq();
System.out.println(x.trim());

Does this line type check

1. Yes
2. No
3. It depends
Subtype Polymorphism

interface ObjQueue {
    public void enq(Object elt);
    public Object deq();
    public boolean isEmpty();
}

ObjQueue q = ...;
q.enq(" CIS 120 ");
Object x = q.deq();
//System.out.println(x.trim());
q.enq(new Point(0.0,0.0));
___B___ y = q.deq();

What type for B?
1. Point
2. Object
3. ObjQueue
4. None of the above
Parametric Polymorphism (a.k.a. Generics)

• Big idea:
  
  Parameterize a type (i.e. interface or class) by another type.

```java
public interface Queue<E> {
    public void enq(E o);
    public E deq();
    public boolean isEmpty();
}
```

• The implementations of a parametric polymorphic interface can not depend on the implementation details of the parameter.
  – e.g. the implementation of enq should not invoke methods on ‘o’
Generics (Parametric Polymorphism)

```java
public interface Queue<E> {
    public void enq(E o);
    public E deq();
    public boolean isEmpty();
    ...
}

Queue<String> q = ...;
q.enq(" CIS 120 ");
String x = q.deq(); // What type of x? String
System.out.println(x.trim()); // Is this valid? Yes!
q.enq(new Point(0.0,0.0)); // Is this valid? No!
```
Subtyping and Generics
Subtyping and Generics*

• Java generics are *invariant*:
  - Subtyping of *arguments* to generic types does not imply subtyping between the instantiations:

  ```java
  Queue<String> qs = new QueueImpl<String>();
  Queue<Object> qo = qs;
  qo.enq(new Object());
  String s = qs.deq();
  ```

* Subtyping and generics interact in other ways too. Java supports “bounded” polymorphism and wildcard types, but those are beyond the scope of CIS 120.
Subtyping and Generics

Which of these are true, assuming that class QueueImpl<E> implements interface Queue<E>?

1. QueueImpl<Queue<String>> is a subtype of Queue<Queue<String>>
2. Queue<QueueImpl<String>> is a subtype of Queue<Queue<String>>
3. Both
4. Neither
The Java Collections Library

A case study in subtyping and generics

(Also very useful!)
Java Packages

• Java code can be organized into *packages* that provide namespace management.
  – Somewhat like OCaml’s modules
  – Packages contain groups of related classes and interfaces.
  – Packages are organized hierarchically in a way that mimics the file system’s directory structure.

• A .java file can *import* (parts of) packages that it needs access to:

```java
import org.junit.Test;    // just the JUnit Test class
import java.util.*;      // everything in java.util
```

• Important packages:
  – java.lang, java.io, java.util, java.math, org.junit

• See documentation at:
  http://docs.oracle.com/javase/7/docs/api/
Reading Java Docs

java.util

https://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html
Interfaces* of the Collections Library

*not all of them!
We’ve already seen this interface in the OCaml part of the course.

Most collections are designed to be mutable (like queues)

* Why not E? Internally, collections use the equals method to check for equality – membership is determined by o.equals, which does not have to be false for objects of different types. Most applications only store and remove one type of element in a collection, in which case this subtlety never becomes an issue.
Sequences

- Iterable\(<E>\>
- Collection\(<E>\>
  - List\(<E>\>
  - Deque\(<E>\>
    - LinkedList\(<E>\>
    - ArrayList\(<E>\>
    - ArrayDeque\(<E>\>

Relationships:
- Extends
- Implements
Sets and Maps*

- Collection\(<E>\)
  - Set\(<E>\)
    - HashSet\(<E>\)
    - TreeSet\(<E>\)
    - SortedSet\(<E>\)
  - TreeSet\(<E>\)
- Map\(<K,V>\)
  - HashMap\(<E>\)
  - SortedMap\(<K,V>\)

*Read javadocs before instantiating these classes! There are some important details to be aware of to use them correctly.
implement Comparable when using SortedSets and Sorted Maps