CS 211: Generics

Chris Kauffman

Week 10

Front Matter

Goals Today

- Generics: Classes with Type Parameters
- ArrayLists and Implementation

Immediate Reading

Lab Manual Ch 17: Generics

Later Reading

- BJP Ch 10.1: Using ArrayList
- BJP Ch 15: Implementing ArrayList

Upcoming

- P4 Due Sunday: Questions?
- Lab 10: Task

Schedule

4/3	Mon	Generics/Exceptions
4/5	Wed	Generics
4/5-7	W-F	Lab 10 Task
4/9	Sun	P4 Due
4/10	Mon	Review
4/12	Wed	Exam 2

Generics: Consider the Array

It's cool: make one of any type

```
String [] sa = ...;
int [] ia = ...;
Rabbits [] ra = ...;
```

ArrayList has *almost* the same ability

```
ArrayList<String> sal = ...;
ArrayList<int> ial = ...;
ArrayList<Rabbits> ra = ...;
```

Which one is wrong?

Type parameters

Arrays

- Arrays are a container, hold stuff
- Can hold any type of thing, parameterized type:
- Not just an array, a String array
- Create/Assign/Access semantics independent content types
- Algorithms based on these semantics won't change either

Exercise: Pair Class

- Attempt to define a `Pair` class to hold pairs of "stuff"
- What solutions can you come up with and what are their limitations?
- How would your solution work with
  - A pair of `Strings`
  - A pair of `Integers`
  - A pair of `Coords`

Defining Generic Classes

- Angle brackets set up type parameters
- A compile-time variable to hold a type
- Class definitions <T> or <K,V> or whatever
- Can also be associatd with methods
- Use type param T or K or whatever wherever unknown type exists
- T specified at use time as String or Integer or whatever

```
public class Pair<T>{
  private T first;
  private T second;

  public Pair(T f, T s){
    this.first = f;
    this.second = s;
  }

  public T getFirst(){
    return this.first;
  }

  public T getSecond(){
    return this.second;
  }

  String toString(){
    return
      String.format("(%s,%s)",
                    first,second);
  }
}
```

Using Generic Classes

I want a Pair of Strings and Pair of Doubles

```
Pair<String> ps = new Pair<String>("spike","faye");
Pair<Double> pd = new Pair<Double>(1.23, 4.56);
String s = ps.getFirst();
Double x = pd.get-Second();

Double y = ps.getSecond(); // ?
```

- Pairs are parameterized on a type
- Pairs are defined in Pairs.java

Want a mixed pair? Define a MixedPair

```
MixedPair<String,Integer> si =
  new MixedPair<String,Integer>("jet",42);
String s = si.getFirst();
Integer i = si.getSecond();
```

Exercise: Generic Construction

How do I declare an `ArrayList` which holds

- Integers
- Coords
- `Pairs` of Integers
- `MixedPairs` of Strings and Coords

How do I declare a ..

- Pair of String ArrayLists
- MixedPair of Integer ArrayList and Coord ArrayList
- A Pair of ArrayLists holding MixedPairs of (Integer,String)

Restriction on Generics: Boxed v Unboxed

Primitives not allowed with Generics

- Generics require a uniform memory model
- Restricted Reference Types Only

Compile errors

```
ArrayList<int> al = new ArrayList<int>();
MixedPair<int,double> p =
  new MixedPair<int,double>(1,3.3);
```

Works Fine

```
ArrayList<Integer> al = new ArrayList<Integer>();
MixedPair<Integer,Double> p =
  new MixedPair<Integer,Double>(1,3.3);
```

How is this alternative different under the hood?

Auto-pugilisim (Autoboxing)

Every primitive has a reference equivalent

Primitive	Reference
int	Integer
double	Double
char	Character
boolean	Boolean

- Java compiler is reasonably smart
- Will insert code to convert between them
- Spares us much suffering

See `AutoPugilism.java`

Trouble with Refs

Easy to get twisted around with interchanging primitives and their bigger siblings, particularly wrt to equality

- Primitives work with ==
- References don't (always)

```
public static void demo1(){
  int i=1, j=1;
  System.out.println(i==j);  // True of False?

  Integer w=new Integer(i), z=new Integer(j);
  System.out.println(w==z);  // True of False?

  Integer x=i, y=j;
  System.out.println(x==y);  // True of False?
}
```

See: `TroubleWithRefs.java` for other twisted conversions

Generics are Complex

Skipping lots of semi-interesting stuff
"I can hold subclasses of generic type E"

```
Collection<? extends E>
```

"I work with a superclass of generic type E"

```
Comparator<? super E>
```

- Mostly these are to satisfy the compiler
- I fiddle with types until javac shuts up
  - (Type theorists cringe, all four of them)