Lists

Discussion F

Generics Tutorial
(Ref: Gilad Baracha)

- Auto Boxing
- Erasure
- Sub Classes
- Unchecked Type cast
- Bounded Wildcards

Iterator

```
List<Integer> lint = new ArrayList<Integer>();  //1
lint.add(new Integer(100));                     //2

Iterator<Integer> intIter = lint.iterator();    //3
Integer i1 = intIter.next();                    //4

lint.add(101);                                  //5
int i2 = intIter.next();                        //6
ConcurrentModificationException
```

Erasure

```
public class X { public X();}                          //7
public class Y extends X { public Y();}                //8

List<X> lx = new ArrayList<X>();                       //9
List<Y> ly = new ArrayList<Y>();                       //10

System.out.println(lx.getClass()==ly.getClass());//12

true
```

Erasure (2)

```
public class X { public X();}                          //7
public class Y extends X { public Y();}                //8

List<X> lx = new ArrayList<X>();                       //9
List<Y> ly = new ArrayList<Y>();                       //10

if (lx instanceof ArrayList<X>) {}//unchecked warning

List<X> otherlx = (List<X>) lx;
```

Subtype

```
public class X { public X();}                          //7
public class Y extends X { public Y();}                //8

List<X> lx = new ArrayList<X>();                       //9
List<Y> ly = new ArrayList<Y>();                       //10
lx = ly;                                               //11
```

Compiler Error

Subtype (2)

```
void printCollection(Collection c) {                    //13
  Iterator i = c.iterator();
  while(i.hasNext()) {
    System.out.println(i.next());
  }
}

void printCollection(Collection<Object> c) {    //14
  for (Object o: c) {
   System.out.println(o);
  }
}

printCollection(lint);                    //??
```

Bounded Wildcards

- Used to express polymorphism
- Wildcard, means some type
  - **<?>**
- lower bound, some super type of T
  - **<? super T>**
- upper bound, some sub class of B
  - **<? extends B>**

Wildcards

```
void printCollection(Collection<?> c){
   for (Object o: c) {
    System.out.println(o);
   }
}

Collection<?> c = new ArrayList<Integer>();
c.add(new Object());    // Error

void printCollection(Collection<? extends Shape> c){
   for (Shape s: c) {
    System.out.println(s.area());
   }
}
```

Generic Functions

```
<T> void arrayToCollection(T[] a, Collection<T> c) {
  for (T t: a) {
    c.add(t);
  }
}

Integer [] a = new Integer[10];...
Collection<Integer> c = new ArrayList<Integer>();
arrayToCollection(a, c);
```

- No need to pass actual type, compiler infers it
- They can have bounds too
- Used when arguments/return type are correlated

Inner Classes

- Block Scoped
  - definition similar to definition of a field or a method
- Static
  - do not have the block scope

Snippets // 1

```
public class X {                                        //1
    private int fx;

    public class I {
      int fi;
      public I() { fi = fx; }
    }
}

X.I ci = new X.I();
java X$I                    // to run main method in X.I
```

// 2

```
public class X {                                        //2
  private int fx;
  private I i = new I();
  private int fx2 = i.fi;

  public class I {
    private int fi;
    public I() { fi = fx; }
  }
}
```

// 3

```
public class X {                                        //3
     int fx;

     public static class I {
       int fi;
       public I() { fi = fx; }
     }
}
```

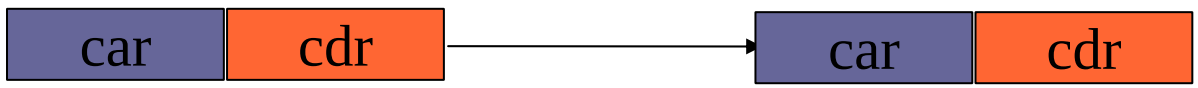
DisInherit Methods

- Not really, has to satisfy Is-a
- Illegal to narrow visibility
- Can reimplement and throw exception

DisInherit

```
public class X {
  public void function() {}
}

public class Y extends X {
  private void function() {}   // compiler error
}


public class Z extends X {
  private void function() {
   throw new NoSuchMethodException();
  }
}
```

Lisp List



Lisp List Interface

```
public interface LispList<E> {

  public int length();
  public E first ();
  public LispList<E> rest ();

}
```

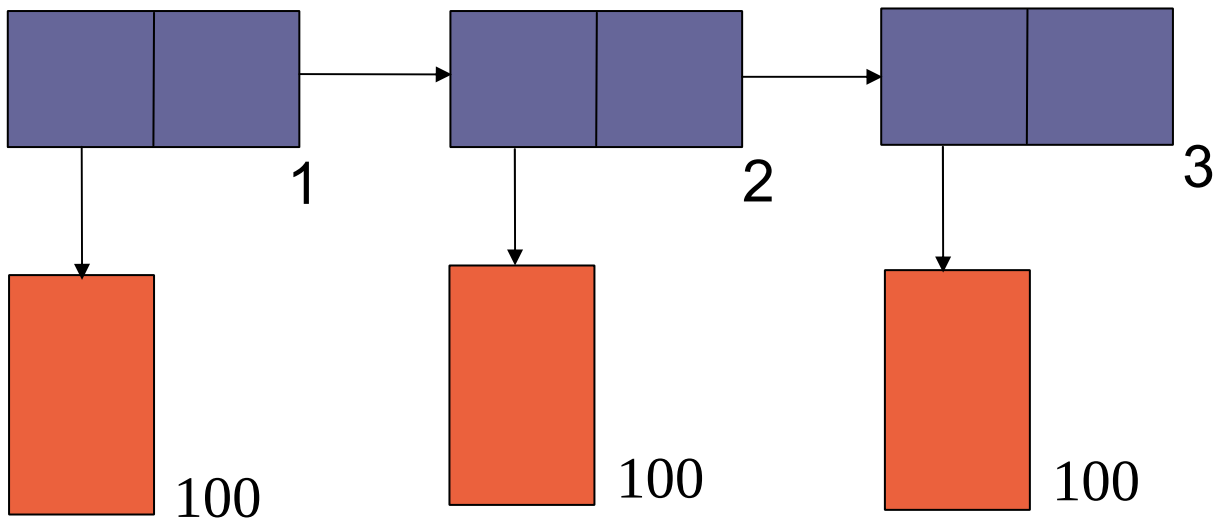
Lisp List

```
public Cons<E> implements LispList<E> {

  E car;
  LispList<E> cdr;

  public Cons(E e) {car = e;}
  public Cons(LispList<E> onelist,
              LispList<E> otherlist) {}

  public int length() {}
  public E first () {return car;}
  public LispList<E> rest () {return cdr;}

}
```

Lisp List

- Empty list
- How to construct from two lists?
- How to find length?

List of Arrays



List Of Arrays

- Each node has same capacity
  - can compute how many nodes from start the required index will be
- nodes have different capacity
  - can balance the list length

Using List<E>

```
public class ListOfArrays<E> {

  int totalCount;
  int totalCapacity;
  int nodeCapacity;
  List<E []> list;

  private void checkIndex(int idx) {}
  public E get(int idx){}
  public void set(int idx, E e) {}

}
```

List of Arrays

```
public class ListOfArrays<E> {

  int totalCount;
  int totalCapacity;
  List<DataNode<E>> list;

  private class DataNode<E> {
    int count;
    int capacity;
    E [] array;
  }
  private void checkIndex(int idx) {}
  public E get(int idx){}
  public void set(int idx, E e) {}
}
```