



# Generics

# Remember “Don’t repeat yourself” ?

- Some code is so generic, that we want to be able to use it on any kind of objects. E.g., ArrayLists

- One way to do this is to use the Object base class

- All classes in Java inherit from Object

- Object obj = new Wolf();

*Object [] array = new Object[]  
array[0] = wolf;*

- But requires run-time casts to get back the object

- Wolf wolf = (Wolf) obj;

- Type safety can’t be verified at compile time

- Generally considered “yucky”

# Side note: Boxing & Auto-boxing

- Primitives (e.g., ints) aren't objects
- To allow them to use things objects can use, we “box” them
  - E.g., the Integer class
    - Integer myInt = new Integer(8);
    - Is a full class object, with more features & overhead
- In some instances, Java does this boxing for you
  - List<Integer> myList = new ArrayList<>();
  - myList.add(8); // the 8 is auto-box'ed
  - int myInt = myList.get(0); // auto-unbox'ed

# Java 5 introduces Generics

- Common programming language feature
- Compiler takes a “generic” version of the code, and generates the specific versions that are needed.
- Key benefit: Type safety
  - E.g., no run-time casting
  - Compiler can find errors, avoid debugging at run time
- `List<Integer> myList = new ArrayList<>();`

# Writing your own generic types

```
public class Box<T> {  
    private List<T> contents;  
  
    public Box() {  
        contents = new ArrayList<T>();  
    }  
  
    public void add(T thing) { contents.add(thing); }  
  
    public T grab() {  
        if (contents.size() > 0) return contents.remove(0);  
        else return null;  
    }  
}
```

parameterized on T

Box < T, E, X >

- Sun's recommendation is to use single capital letters (such as T) for types
- If you have more than a couple generic types, though, you should use better names

# Methods can be generic

- Even if the class isn't.
- Need enough information to distinguish which version to call

- `public <T> T foo(T in) {  
 return in;  
}`

`foo(11)`

`foo("blah")`

- Not:

- `public <T> T foo(int in) {  
 ...  
}`

`Integer myInt = foo();  
String str = foo();`

# Comparable

- Standard Java interface for comparing things:
  - Provides the compareTo method

~~Box<W>~~

```
public class Box<T> extends Comparable<T>>
```

```
Double d1 = 7;  
Double d2 = 11;  
d1.compareTo(d2) // return negative #
```