

### References

- Textbook (Weiss), sec. 1.5.3
- Sun online Java tutorial
   java.sun.com/docs/books/tutorial/extra/generics/index.html
- For the truly hard-core: Java Generics and Collections, Maurice Naftalin & Philip Wadler, O'Reilly, 2006

The Java Programming Language, 4th ed., Arnold, Gosling & Holmes, A-W, 2006

 And for the Language Lawyers in the crowd: *The Java Language Specification*, 3<sup>rd</sup> ed., Gosling, Joy, Steele & Bracha, A-W, 2005

2

4

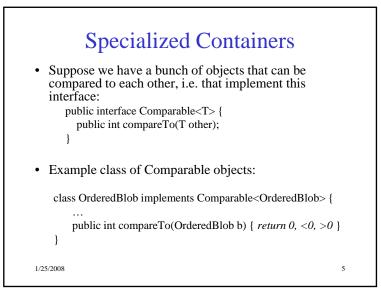
1/25/2008

# function of the provide the provide

### Why?

- Main advantage is compile-time type checking:
  - Ensure at compile time that items put in a generic container have the right type
  - No need for a cast to check the types of items returned; guaranteed by type system
- Underneath, everything is a raw object, but we don't have to write the casts explicitly or worry about type failures

1/25/2008



### Container for Comparable Things

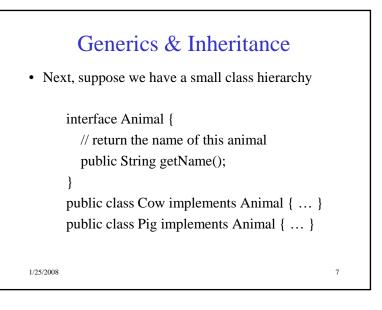
• Suppose we want a container that only holds objects that are Comparable. Here's how:

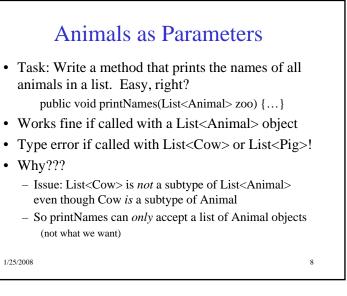
interface SortedCollection <E extends Comparable<E>>

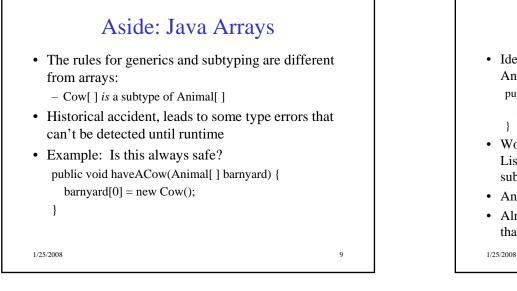
- E must be some type that "extends" (i.e., implements) Comparable<E>
- : can use CompareTo(E) in implementation
- This isn't quite general enough, but it's in the right direction

6

1/25/2008







# **Bounded Wildcards**

- Idea: specify that the parameter can be a list of either Animals or any of Animal's subtypes public void printNames (List<? extends Animal> zoo) { for (Animal a: zoo) System.out.println(a.getName());
- Works great. This is a bounded wildcard. Any List<*t*> works provided that *t* is Animal or some subtype of Animal
- Animal is an upper bound for the wildcard
- Almost always what you want if a method argument that you read from has a parameterized type

### Lower Bounds

- There is corresponding syntax for lower bounds: public void haveACow(List<? super Cow> barnyard) { barnyard.add(new Cow()); // OK
- This is also a wildcard type where Cow is a *lower bound*. Actual argument can be List<Cow>. List<Animal>, List<Object> or any other List whose elements are supertypes of Cow.
  - But not List<Pig>
- Almost always what you want if a method stores into an argument that has a parameterized type

1/25/2008

11

### **Constraints Revisited**

• Recall the type declaration for collection of Comparable objects: interface SortedCollection <E extends Comparable<E>> • Works, but is too restrictive. It requires that E directly implement Comparable<E>, but that's not the only way two E objects can be Comparable. • Solution: interface SortedCollection <E extends Comparable<? super E>> - Can compare two elements of type E as long as E extends Comparable<T> where T is any supertype of E 12

1/25/2008

10



- Type parameters are a compile-time-only artifact. At runtime, only the raw types are present
- So, at runtime, the compile-time class Bag<E> is just a Bag (only one instance of class Bag), and everything added or removed is just an Object, not a particular E
  - Casts, etc. are inserted by compiler as needed, but guaranteed to succeed if generics rules are obeyed
  - Underlying code and JVM is pre-generics Java
- Ugly, but necessary design decision
  - Makes it possible for new code that uses generics to interoperate with old code that doesn't
  - Not how you would do it if you could start over

1/25/2008

# Type Erasure Consequences • Code in a class cannot depend on the actual value of a type parameter at runtime. Examples of problems: public class Bag<E> { public static E makeE() { ... } // error – what is E? private E oneE; // OK private E[] arrayE; // also OK public void makeStuff() { oneE = new E(); // error – new E() not allowed arrayE = new E[]; // error – new E[] also not allowed } } }

1/25/2008

## But I Need to Make an E[]!!!!

- Various solutions. For simple case, we can use an unchecked cast of an Object array (which is what it really is underneath anyway)
  - E[] stuff = (E[])new Object[size];
  - All the other code that uses stuff[] and its elements will work and typecheck just fine
- Be sure you understand the cause of *all* unchecked cast warnings, & limit to "safe" situations like this
- More complex solutions if you want more type safety or have more general requirements see references for detailed discussions

1/25/2008

15

13

# Example with "Generic" Array

14

public class Bag <e> {</e>	// methods
// instance variable	public void store(E item)
E[] items;	{ items[0] = item; }
// constructor	public E get()
<pre>public Bag() {</pre>	{ return items[0]; }
items = $(E[ ])$ new	
Object[10];	}
}	