
artificially flavored root beer
artificially flavored strawberry soda
cola
artificially flavored grape soda
cola
NET 12 FL OZ
CHERRY COLA
ROOT BEER
12 FL OZ (355 ML)
LEMON-LIME SODA
soda

# Generics

# Generic

- Definition: characteristic of or relating to a class or group of things; not specific

- Computer Science: a style of computer programming in which algorithms are written in terms of types to-be-specified-later (Wikipedia Generic Programming)

- Introduced to Java in Java 5.0 (2004)
  - Generic Classes (and Interfaces)
  - Generic Methods (and constructors)

# Simple Class Generic

```
public class LinkedList<E> {
    private E payload;
    private LinkedList<E> next;
    public LinkedList(E payload) { this.payload=payload; }
    public void insertTail(E obj) {
        if (next!=null) next.insertTail(obj); else next=new LinkedList<E>(obj);
    }
    public static void main(String[] args) {
        LinkedList<String> list = new LinkedList<String>(args[0]);
        for(int i=1;i<args.length;i++) list.insertTail(args[i]);
    }
}
```

Angle Brackets enclose type variable
By convention, type variables are a single, upper case letter

The rest of the class uses "E" as a type

Type of reference variable specifies real type

Constructor invocation specifies real type

method arguments type-checked

# Generic Implementation Strategy

**C++ (Template style)**

- At compile time, create multiple classes (or functions); one for each type used
- List<E> class becomes
  - List<Integer>
  - List<String>
  - etc.

**Java (Type-Checked Generic)**

- Compiler checks type safety
- "Erasure"…
  - Compiler ensures code works without knowing the type
- Single class/method!

# Erasure… Compiler checks…

```
public class LinkedList {
        private X payload;
        private LinkedList next;
        public LinkedList(X payload) { this.payload=payload; }
        public void insertTail(X obj) {
                if (next!=null) next.insertTail(obj); else next=new LinkedList(obj);
        }
        …
}
```

Does this code work for any X?

Assign X to X- OK

X parameter to insertTail - OK

X parameter to constructor- OK

Everything works… at run-time just run this code

# Generics and Primitive Types

- Primitive types (int, float, double, char, …) are not allowed when instantiating a generic.
- Use the primitive box classes (Integer, Double, etc.) instead
- There is a Number class in Java
  - Superclass of all numeric box classes
  - Has methods like intValue() and doubleValue()

# Warning: Generics of SubClasses

- Generics of Sub-Classes are **not** sub-classes!

```
public class Bird { … }
public class Duck extends Bird { … }

LinkedList<Bird> blist = new LinkedList<Bird>(new Duck(…));
LinkedList<Duck> dlist = new LinkedList<Duck>(new Duck(…));
LinkedList<Bird> blist2 = dlist;
```

I can put a Duck in a BirdList

Type mismatch: cannot convert from LinkedList<Duck> to LinkedList<Bird>
because LinkedList<Duck> is not a subclass of LinkedList<Bird>
(If this were allowed, I could add a Bird to blist2, which is an alias for dlist
But I shouldn't allow Birds in a Duck list!)

# Wildcards and Upper/Lower Bounds

- ? in generic means "any class"
- Upper Bound: <? extends E> means E, or any sub-class of E
- Lower Bound: <? super E> means E, or any super-class of E

Allows either LinkedList<Bird> or LinkedList<Duck> to be added to LinkedList<Bird>

```
public void addList(LinkedList<? extends E> app) {
    if (app!=null) { insertTail(app.payload); addList(app.next); }
}
```

# Method Generics

- Syntax:
  *attributes* <*spec*> *returnType name*(*parameter_list*) {…
- For example:

```
public static <T extends Number> double sum(T … vals) {
    double sum=0.0;
    for(T val : vals) { sum+=val.toDouble(); }
    return sum;
}
```