

CMSC 132: OBJECT-ORIENTED PROGRAMMING II



Generic Programming

Department of Computer Science
University of Maryland, College Park

Generic Programming

- Generic programming
 - Defining constructs that can be used with different data types
 - I.e., using same code for different data types
 - Example: stack operations the same regardless of stack element type
- Implemented in Java through
 - Inheritance → A extends B
 - Type variables → <A>

Generic Programming Examples

- **Inheritance**

```
Class A {  
    doWork( A x ) { ... }  
}  
Class B extends A { ... }
```

```
A w1 = new A( );  
B w2 = new B( );
```

```
w1.doWork( w1 );  
w2.doWork( w2 );
```



**doWork() applied to objects of both
class A and B**

- **Type Variables**

```
Class W<T> {  
    doWork( T x ) { ... }  
}  
Class A { ... }  
Class B { ... }
```

```
W<A> x1 = new W<A>( );  
W<B> x2 = new W<B>( );  
A w1 = new A( );  
B w2 = new B( );
```

```
x1.doWork( w1 );  
x2.doWork( w2 );
```

Generic Class

- Class with one or more type variables
 - Example → `class ArrayList<E>`
- To use generic class, provide an actual type
 - Valid types
 - Class → `ArrayList<String>`
 - Interface → `ArrayList<Comparable>`
 - Invalid types
 - Primitive type → `ArrayList<int>`
(use wrappers) → `ArrayList<Integer>`

Defining a Generic Class

- Example

```
public class myGeneric<T> {  
    private T value;  
    public myGeneric( T v ) { value = v; }  
    public T getVal( ) { return value; }  
}
```

- Append type variable(s) to class name using angle brackets **ClassName<type variable>**
- Can use any name for type variable (but typically single uppercase letter → E, K, V, etc...)
 - <http://download.oracle.com/javase/6/docs/api/java/util/Map.html>
- Use the type variable to define type of variables, type of method parameters, method return type and object allocation
- Arrays
 - Type of an array object may not be a type variable or a parameterized type, unless it is an unbounded wildcard type
 - How to define arrays?
 - `T[] data = (T[]) new Object[size];`
- **Example:** Queue.java

Generics and Subtyping

- In general if B is a subtype of A, and GT is a generic type declaration, it is not the case that GT is a subtype of GT<A>
- **Example**

```
ArrayList<String> strL = new ArrayList<String>();  
ArrayList<Object> objL = strL; // Illegal!
```

Generics and Subtyping

- Consider what could happen if legal

```
class A { ... }
```

```
class B extends A { ... } // B is subtype of A
```

```
List<B> bL = new ArrayList<B>();
```

```
List<A> aL = bL;
```

```
aL.add(new A());
```

```
B b = bL.get(0); // runtime exception
```

- Using String Class

```
ArrayList<String> sL = new ArrayList<String>();
```

```
ArrayList<Object> oL = sL; // Illegal, but let's assume is valid
```

```
oL.add(new Integer(10));
```

```
String entry = sL.get(0); // Problem!!
```

Subtyping and Arrays

- Subtyping works for arrays

```
class A { ... }
```

```
class B extends A { ... } // B is subtype of A
```

```
A a = new B(); // B can be used where A expected
```

```
B[] bB = new B[1];
```

```
A[] aB = bB;
```

```
bB[0] = a; // won't compile
```

- Using String Class

```
Object value = new String("HI");
```

```
String[] sS = new String[1];
```

```
Object[] oO = sS; // Legal
```

```
sS[0] = value; // It will not Compile
```

- **Example:** Fruit.java, TropicalFruit.java

Wildcards

- ? (unknown)
 - Collection<?>
 - Collection whose element type matches anything
- Bounded Wildcard
 - Example: ArrayList<? extends Shape>
 - Unknown type that is Shape or subtype of Shape
 - Notice the meaning of extends in this context
- Summary
 - `<?>` → unknown type
 - `<? extends typeExpression>` → unknown type that is *typeExpression* or a subtype of typeExpression
 - `<? super typeExpression>` → unknown type that is *typeExpression* or a supertype of *typeExpression*.
 - *typeExpression* can involve further occurrences of wildcard type expressions
- **Example:** WildCard.java