CS 240

Data Structures and Algorithms I

Alex Vondrak

ajvondrak@csupomona.edu

October 21, 2011

A Note About Project 1

Use this:

```
class StackUnderflowException extends Exception {
    public StackUnderflowException() {
        super("Stack underflow.");
    }
}
```

Generalizing Stacks—First Try

```
interface Stack {
    public void push(Object value);

    public Object pop()
        throws StackUnderflowException;

    public Object top()
        throws StackUnderflowException;

    public boolean isEmpty();

    public int size();
}
```

Generics

- A generic method depends on an unspecified underlying data type
- A generic class allows us to leave a data type unspecified across the whole class

Syntax

Instead of writing just

```
class Foo {
    // ...
}
```

we may provide a generic type parameter, like

```
class Foo<E> {
    // ...
}
```

Generics

Instantiation

When instantiating a generic class, we must say what its generic type parameter will be, like

```
Foo<Integer> x = new Foo<Integer>();
Foo<Boolean> y = new Foo<Boolean>();
Foo<String>  z = new Foo<String> ();
```

Restriction

Generic type parameters must always be instantiated with class types.

Generics

What Do They Buy Us?

Generic type parameters can be used as "type variables" within the class

Before

```
class Foo {
    public Object bar() {
        // ...
    }
}

Foo x = new Foo();
Integer baz = (Integer) x.bar();
```

Generics

What Do They Buy Us?

Generic type parameters can be used as "type variables" within the class

After

```
class Foo <E> {
    public E bar() {
        // ...
    }
}

Foo<Integer> x = new Foo<Integer>();
Integer baz = x.bar();
```

Generics

Restrictions

- Cannot call the constructor of a generic type

```
class Foo<E> {
    public Foo() {
        E someObject = new E(x, y, z); ✗
    }
}
```

- Cannot create a new array of a generic type

```
class Foo<E> {
    public Foo() {
        E[] someArray = new E[100]; ✗
    }
}
```

Building A Generic ArrayStack

```
class ArrayStack /////////////Stack {
  public void push(int value) { ... }

  public int pop()
     throws StackUnderflowException { ... }

  public int top()
     throws StackUnderflowException { ... }

  public boolean isEmpty() { ... }

  public int size() { ... }
}
```