Generics

CS 240

Alex Vondrak

ajvondrak@csupomona.edu

Winter 2012

Class Hierarchy

- In Java, classes form a hierarchy by being subclasses of each other
- **`class`** `A` **`extends`** `B` establishes `A` as a subclass of `B`

Is every class a subclass of (or equal to) itself?

(A) Yes

(B) No

Class Hierarchy

- In Java, classes form a hierarchy by being subclasses of each other
- `class A extends B` establishes `A` as a subclass of `B`

Consider two distinct classes, `A` and `B`. Is it possible for both `A` to be a subclass of `B` and `B` to be a subclass of `A`?

(A) Yes

(B) No

Class Hierarchy

- In Java, classes form a hierarchy by being subclasses of each other
- `class A extends B` establishes `A` as a subclass of `B`

Consider three classes such that

- `class A extends B`
- `class B extends C`

Is `A` a subclass of `C`?

(A) Yes

(B) No

Partial Orders

Definition

A partial order is a binary relation "$\sqsubseteq$" over a set P that satisfies the following properties:

- Reflexivity: $\forall a \in P, a \sqsubseteq a$
- Antisymmetry: $\forall a, b \in P, a \neq b \implies a \not\sqsubseteq b \lor b \not\sqsubseteq a$
- Transitivity: $\forall a, b, c \in P, a \sqsubseteq b \land b \sqsubseteq c \implies a \sqsubseteq c$

P is sometimes called a partially-ordered set (or poset)

Definition (Greatest Element)

An element $g \in P$ such that $\forall a \in P, a \sqsubseteq g$.

Definition (Least Element)

An element $l \in P$ such that $\forall a \in P, l \sqsubseteq a$.

Multiple Choice Question

What is the greatest element in the Java class hierarchy?

(A) `Object`

(B) **`null`**

(C) Depends on the class in question

(D) There is none

Multiple Choice Question

What is the least element in the Java class hierarchy?

(A) `Object`

(B) **`null`**

(C) Depends on the class in question

(D) There is none

Multiple Choice Question

Must it be the case that either:

- `A` is a subclass of `B`
- `B` is a subclass of `A`
- `A` is the same class as `B`

for arbitrary Java classes `A` and `B`?

(A) Yes

(B) No

Multiple Choice Question

Can an instance of one class be converted into an instance of another class?

(A) Yes (given certain conditions)

(B) No (never)

Multiple Choice Question

Suppose we have the code

```
String s = "Something";
Object o;

o = s;
```

Is this code valid?

(A) Yes

(B) No

Multiple Choice Question

Suppose we have the code

```
String s = "Something";
Object o;

o = s;
```

When `s` is stored in `o`, the value is converted from a `String` into an `Object`. Which direction is this in the class hierarchy?

(A) Up

(B) Down

Multiple Choice Question

Suppose we have the code

```
String s = "Something";
Object o;
o = s;

s = "Different";
s = o;
```

Is this code valid?

(A) Yes: we are moving up the hierarchy

(B) Yes: Java will convert down the hierarchy

(C) No: we are moving down the hierarchy

(D) No: `String` and `Object` are disjoint classes

Multiple Choice Question

Suppose we have the code

```
String s = "Something";
Object o;
o = s;

s = "Different";
s = (String) o;
```

Why is this code valid?

(A) We are moving up the hierarchy

(B) We are moving down the hierarchy, and have a typecast to the effect

(C) Typecasts allow us to convert between any two classes

(D) None of the above

Narrowing vs Widening Conversions

Definition (Widening Conversion)

In Java, conversions up the class hierarchy (i.e., “widening” the type) are allowed without a problem.

Definition (Narrowing Conversion)

In Java, conversions down the class hierarchy (i.e., “narrowing” the type) may need a typecast to work.

Multiple Choice Question

In Java, is “everything an object”?

(A) Yes

(B) No

Wrapper Classes

- **`boolean`**: `Boolean`
- **`byte`**: `Byte`
- **`char`**: `Character`
- **`double`**: `Double`
- **`float`**: `Float`
- **`int`**: `Integer`
- **`long`**: `Long`
- **`short`**: `Short`

Multiple Choice Question

Suppose we have the following:

```
int i = 42;
int j;
Integer k;
```

Which assignment below is valid?

(A) `k = new Integer(i);`

(B) `k = i;`

(C) Both of the above

(D) Neither of the above

Multiple Choice Question

Suppose we have the following:

```
int i = 42;
int j;
Integer k;

k = new Integer(i);
```

Which assignment below is valid?

(A) `j = k.intValue();`

(B) `j = k;`

(C) Both of the above

(D) Neither of the above

Boxing and Unboxing

Definition

Primitive types can be boxed into their wrapper classes

Definition

Wrapper objects can be unboxed into primitive types

Note

Java generally handles boxing/unboxing automatically

Multiple Choice Question

```
interface Stack {
   public void push(int item);

   public int pop() throws StackUnderflowException;

   public int peek() throws StackUnderflowException;
}
```

If we wanted a stack that could hold arbitrary objects, what type could we use to replace **`int`**?

(A) We can't do that, due to conversions that are neither narrowing nor widening

(B) No need; just make a variable that holds the type

(C) `Object`

(D) `Generic`

Multiple Choice Question

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

   public Object pop() throws StackUnderflowException;

   public Object peek() throws StackUnderflowException;
}
```

Why don't we want to do this?

(A) Every return value would need a typecast

(B) The stack elements might not have the same types

(C) We lose information about what the stack is meant to hold

(D) We can't store primitive data types

Generics

Definition (Generic Method)

A method that depends on an unspecified underlying data type

Definition (Generic Class/Interface)

A generic class (or generic interface) allows us to leave a data type unspecified across the whole class (or interface) by replacing it with a generic type parameter

Before

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

   public Object pop() throws StackUnderflowException;

   public Object peek() throws StackUnderflowException;
}
```

Generics

Definition (Generic Method)

A method that depends on an unspecified underlying data type

Definition (Generic Class/Interface)

A generic class (or generic interface) allows us to leave a data type unspecified across the whole class (or interface) by replacing it with a generic type parameter

After

```
interface Stack<E> {
   public void push(E item);

   public E pop() throws StackUnderflowException;

   public E peek() throws StackUnderflowException;
}
```

Restrictions

- Cannot call the constructor of a generic type
- Cannot create a new array of a generic type
- Generic type parameters must represent classes (not primitive data types)

```
class Foo<E> {
    public Foo() {
        E someObject = new E(x, y, z); ✗
        E[] someArray = new E[100]; ✗
    }
}
⋮
Foo<int> bar = new Foo<int>(); ✗
```

Restrictions

- Cannot call the constructor of a generic type
- Cannot create a new array of a generic type
- Generic type parameters must represent classes (not primitive data types)

```
class Foo<E> {
    public Foo() {
        E someObject = new E(x, y, z); ✗
        E[] someArray = (E[]) new Object[100]; ✓
    }
}
⋮
Foo<Integer> bar = new Foo<Integer>(); ✓
```

Multiple Choice Question

```
class ArrayStack implements Stack {
    private static final int INITIAL_CAPACITY = 10;
    private int[] data;
    private int top;
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. What do you think the **`class`** declaration would be?

(A) `class ArrayStack implements Stack`

(B) `class ArrayStack implements Stack<E>`

(C) `class ArrayStack<E> implements Stack`

(D) `class ArrayStack<E> implements Stack<E>`

Multiple Choice Question

```
class ArrayStack<E> implements Stack<E> {
   private static final int INITIAL_CAPACITY = 10;
   private int[] data;
   private int top;
   ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. Which field's type do we change?

(A) `INITIAL_CAPACITY`

(B) `data`

(C) `top`

(D) More than one of the above

Multiple Choice Question

```
class ArrayStack<E> implements Stack<E> {
    private static final int INITIAL_CAPACITY = 10;
    private int[] data;
    private int top;
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. What should data's type be?

(A) `private int[] data`

(B) `private Object[] data`

(C) `private E[] data`

(D) `private E data`

Multiple Choice Question

```
    ⋮
public ArrayStack() {
    this.data = new int[this.INITIAL_CAPACITY];
    this.top = -1;
}
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should we initialize **`this`**`.data` now?

(A) `this.data = new E[this.INITIAL_CAPACITY];`

(B) `this.data = new Object[this.INITIAL_CAPACITY];`

(C) `this.data = (E[]) new Object[this.INITIAL_CAPACITY];`

(D) `this.data = (Object[]) new E[this.INITIAL_CAPACITY];`

Multiple Choice Question

```
    ⋮
public int size() {
    return this.top + 1;
}

public boolean isEmpty() {
    return this.size() == 0;
}
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should the types of `size` and `isEmpty` change?

(A) They shouldn't

(B) `size` should return `E`

(C) `size` should return `Integer`

(D) `isEmpty` should return `E`

Multiple Choice Question

```
    ⋮
public int peek() throws StackUnderflowException {
    if (this.isEmpty()) {
        throw new StackUnderflowException();
    }
    return this.data[this.top];
}
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should peek change?

(A) It shouldn't

(B) It should return `E`

(C) We need some typecasting logic in the **`return`**

(D) `StackUnderflowException` should be generic

Multiple Choice Question

```
   ⋮
public int pop() throws StackUnderflowException {
   int result = this.peek();
   this.top--;
   return result;
}
   ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should pop change?

(A) It should return `E`

(B) We should **null** out **this**.data[**this**.top]

(C) Both of the above

(D) None of the above

Multiple Choice Question

```
    ⋮
public void push(int item) {
    if (this.size() == this.data.length) {
        this.grow();
    }
    this.top++;
    this.data[this.top] = item;
}
    ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should push change?

(A) It shouldn't

(B) It should take in an `E item`

(C) We should **`null`** out `this.data[this.top]` before storing `item`

(D) None of the above

Multiple Choice Question

```
   ⋮
private void grow() {
   int[] biggerArray = new int[2 * this.data.length + 1];
   for (int i = 0; i < this.data.length; i++) {
      biggerArray[i] = this.data[i];
   }
   this.data = biggerArray;
}
   ⋮
```

Suppose we make our `ArrayStack` generic using the `Stack<E>` interface. How should `grow` change?

(A) It should return the type `E`

(B) Instead of **`int`**`[]`, it should use `Integer[]`

(C) Instead of **`int`**`[]`, it should use `E[]`

(D) None of the above