Programming Languages and Techniques (CIS1200)

Lecture 26

Static Methods, Generics

Chapters 24 and 25

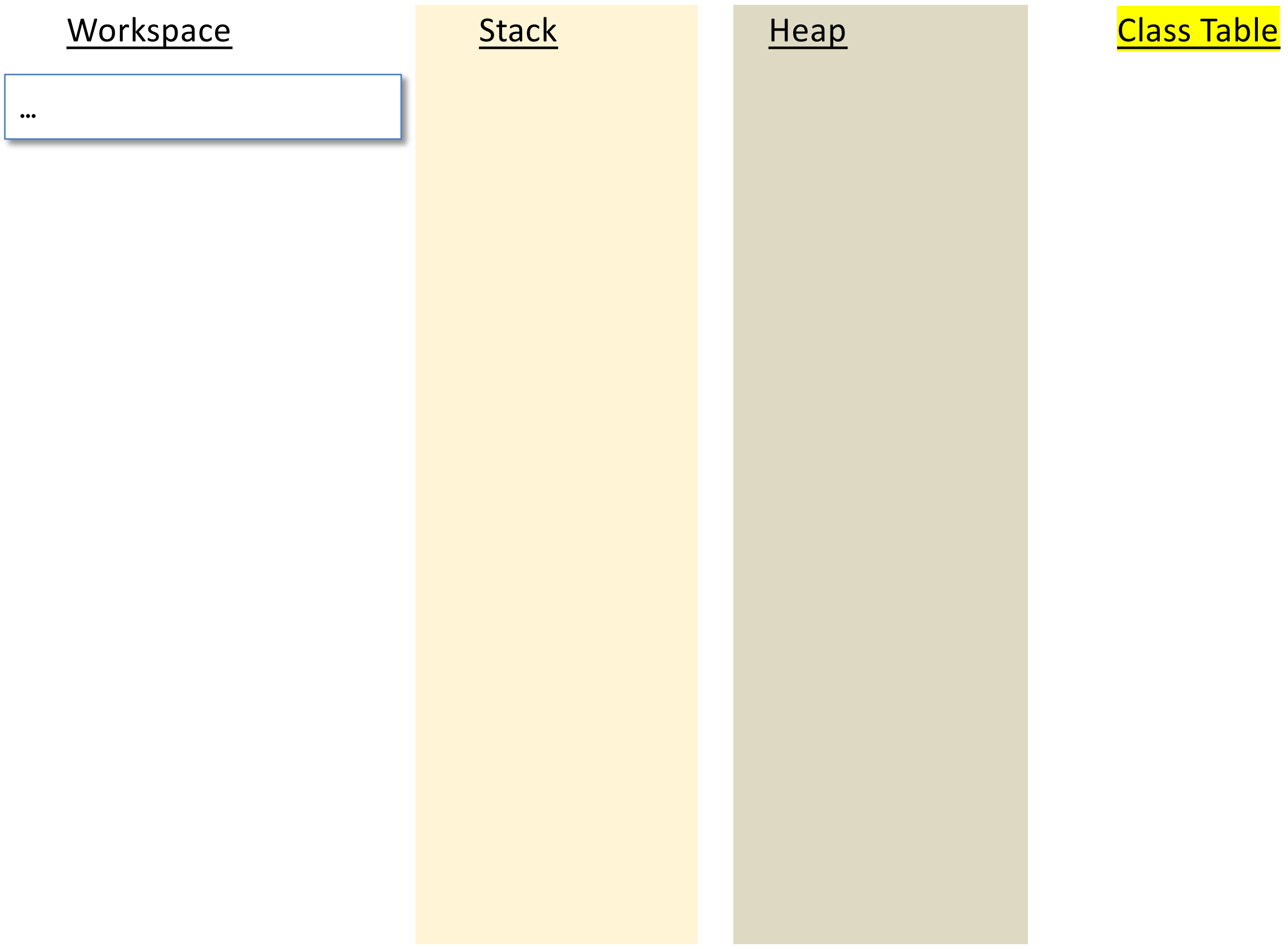
Announcements

- HW07: PennPals
  - Programming with Java Collections
  - Available soon
  - Due Tuesday, April 8 at 11.59pm

Inheritance and Dynamic Dispatch

When do constructors execute?
How are fields accessed?
What code runs in a method call?
What is ‘this’?

ASM refinement: The Class Table



Workspace

…

Stack

Heap

Class Table

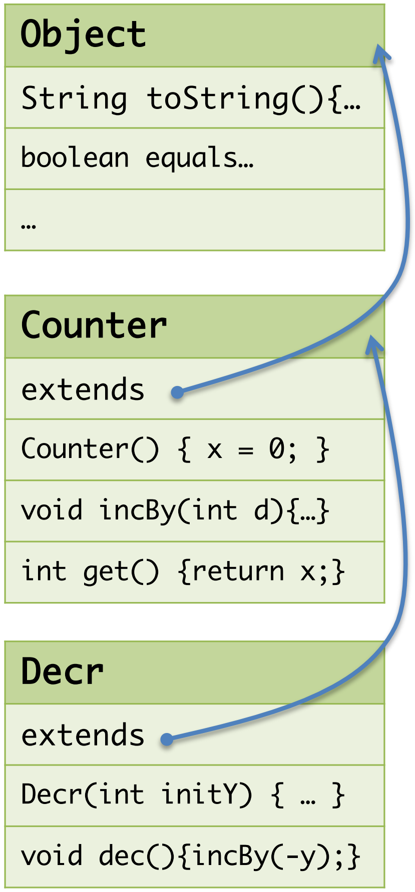
ASM refinement: The Class Table

```
public class Counter {
  private int x;
  public Counter () { x = 0; }
  public void incBy(int d) { x = x + d; }
  public int get() { return x; }
}

public class Decr extends Counter {
  private int y;
  public Decr (int initY) { y = initY; }
  public void dec() { incBy(-y); }
}
```

The class table contains:

- the code for each method,
- references to each class's parent, and
- the class's static members.

Class Table



Object

String toString(){…

boolean equals…

…

Counter

extends

Counter() { x = 0; }

void incBy(int d){…}

int get() {return x;}

Decr

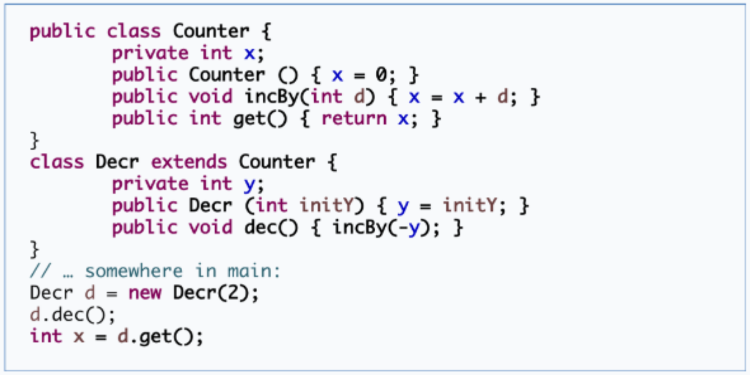
extends

Decr(int initY) { … }

void dec(){incBy(-y);}

26: What is the value of *x* at the end of this computation?

✓✓ 0



```
public class Counter {
        private int x;
        public Counter () { x = 0; }
        public void incBy(int d) { x = x + d; }
        public int get() { return x; }
}
class Decr extends Counter {
        private int y;
        public Decr (int initY) { y = initY; }
        public void dec() { incBy(-y); }
}
// … somewhere in main:
Decr d = new Decr(2);
d.dec();
int x = d.get();
```

-2 0%

-1 0%

0 0%

1 0%

2 0%

NullPointerException 0%

Doesn't type check 0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Inheritance Example

```
public class Counter {
    private int x;
    public Counter () { x = 0; }
    public void incBy(int d) { x = x + d; }
    public int get() { return x; }
}
class Decr extends Counter {
    private int y;
    public Decr (int initY) { y = initY; }
    public void dec() { incBy(-y); }
}
// … somewhere in main:
Decr d = new Decr(2);
d.dec();
int x = d.get();
```

What is the value of x at the end of this computation?

1. -2
2. -1
3. 0
4. 1
5. 2
6. NPE
7. Doesn't type check

Answer: -2

Static members and the Java ASM

Static Members

- Classes in Java can also act as *containers* for code and data.
- The modifier `static` means that the field or method is associated with the class and *not* instances of the class.

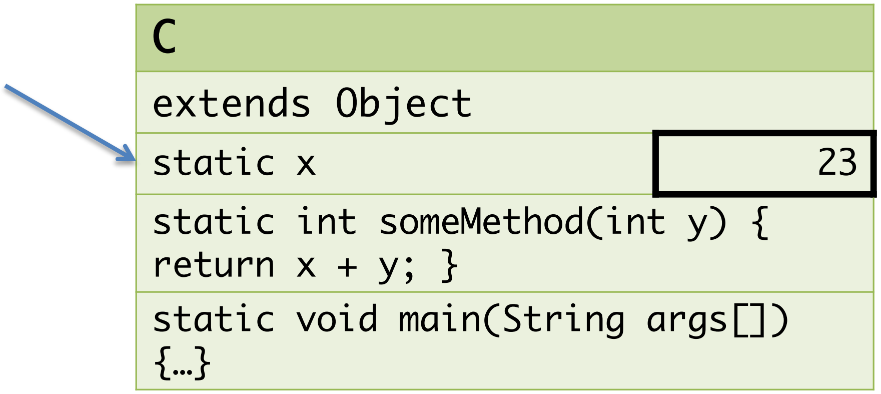
You can do a static assignment to initialize a static field

```
class C {
  public static int x = 23;
  public static int someMethod(int y) { return C.x + y; }
  public static void main(String args[]) {
    …
  }
}

 // Elsewhere:
C.x = C.x + 1;
C.someMethod(17);
```

Access to the static member uses the class name `C.x` or `C.foo()`

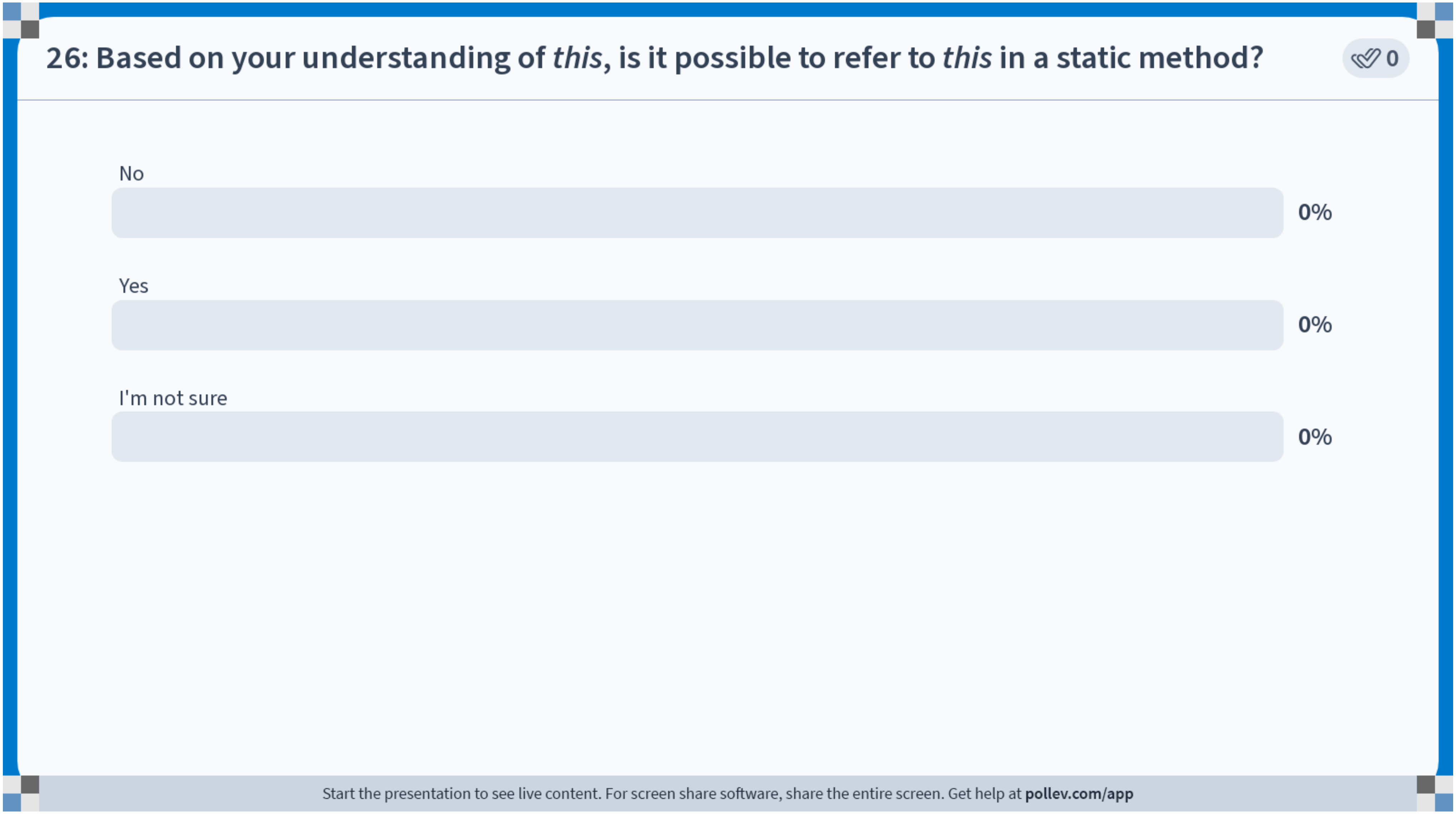
Class Table Associated with C

- The class table entry for C has a field slot for x.
- Updates to C.x modify the contents of this slot: C.x = 17;



```
C
extends Object
static x                          23
static int someMethod(int y) {
return x + y; }
static void main(String args[])
{…}
```

- A static field is a *global* variable
  - There is only one heap location for it (in the class table)
  - Modifications to such a field are visible everywhere the field is
    - if the field is public, this means *everywhere*
  - Use with care!



26: Based on your understanding of *this*, is it possible to refer to *this* in a static method?

✓ 0

No

0%

Yes

0%

I'm not sure

0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Based on your understanding of ‘this’, is it possible to refer to ‘this’ in a static method?

1. No
2. Yes
3. I’m not sure

Static Methods (Details)

- Static methods do *not* have access to a `this` reference
  - Why? There isn't an instance to dispatch through!
  - Therefore, static methods may only directly call other static methods.
  - Similarly, static methods can only directly read/write static fields.
  - Of course a static method can create instance of objects (via `new`) and then invoke methods on those objects.

- Gotcha: It is possible (but confusing) to invoke a static method as though it belongs to an object instance.
  - e.g. `o.someMethod(17)` where `someMethod` is static

Java Generics

Subtype Polymorphism

vs.

Parametric Polymorphism

Review: Subtype Polymorphism*

- Main idea:

> Anywhere an object of type A is needed, an object that is a **subtype** of A can be provided.

- Why is this ok? If B is a subtype of A, it provides all of A's (public) methods.

*polymorphism = many shapes

Is subtype polymorphism enough?

Mutable Queue Interface in OCaml

```
module type QUEUE =
sig
  (* type of the data structure *)
  type 'a queue
  (* Make a new, empty queue *)
  val create : unit -> 'a queue
  (* Add a value to the end of the queue *)
  val enq : 'a -> 'a queue -> unit
  (* Remove the front value and return it (if any) *)
  val deq : 'a queue -> ‘a
  (* Determine if the queue is empty *)
  val is_empty : ‘a queue -> bool
end
```

How can we translate this interface to Java?

Java Interface using Subtyping

```
module type QUEUE =
sig
  type 'a queue

  val create : unit -> 'a queue
  val enq : 'a -> 'a queue -> unit
  val deq : 'a queue -> 'a
  val is_empty : 'a queue -> bool
end
```

OCaml

```
interface ObjQueue {


   // no constructors
   // in an interface
   public void enq(Object elt);
   public Object deq();
   public boolean isEmpty();

}
```

Java

Subtype Polymorphism

```
interface ObjQueue {
   public void enq(Object elt);
   public Object deq();
   public boolean isEmpty();
}
```

```
ObjQueue q = …;

q.enq(" CIS 120 ");
__A__   x = q.deq();
```

What type should we write for A?

1. String
2. Object
3. ObjQueue
4. None of the above

ANSWER: Object

Subtype Polymorphism

```
interface ObjQueue {
    public void enq(Object elt);
    public Object deq();
    public boolean isEmpty();
}
```

```
ObjQueue q = …;

q.enq(" CIS 120 ");
Object  x = q.deq();
System.out.println(x.trim());
```

trim is a method of the String class (removes extra spaces)

← Does this line type check

1. Yes
2. No
3. It depends

ANSWER: No

Subtype Polymorphism

```
interface ObjQueue {
   public void enq(Object elt);
   public Object deq();
   public boolean isEmpty();
}
```

```
ObjQueue q = …;

q.enq(" CIS 120 ");
Object  x = q.deq();
//System.out.println(x.trim());
q.enq(new Point(0.0,0.0));
___B___  y = q.deq();
```

What type for B?

1. Point
2. Object
3. ObjQueue
4. None of the above

ANSWER: Object

Parametric Polymorphism (a.k.a. Generics)

- Main idea:

Parameterize a type (i.e. interface or class) by another type.

```
public interface Queue<E> {
  void enq(E o);
  E deq();
  boolean isEmpty();
}
```

- Any implementation of the generic interface *cannot* depend on the implementation details of the parameter E.
  - i.e., the implementation of enq cannot invoke any methods on ‘o’ (except those inherited from Object)
  - i.e., the only thing we know about E is that it is a subtype of Object

Generics (Parametric Polymorphism)

```
public interface Queue<E> {
  void enq(E o);
  E deq();
  boolean isEmpty();
  …
}
```

```
Queue<String> q = …;

q.enq(" CIS 120 ");
String x = q.deq();              // What type of x?   String
System.out.println(x.trim());    // Is this valid?    Yes!
q.enq(new Point(0.0,0.0));       // Is this valid?    No!
```

Subtyping and Generics

Subtyping and Generics*

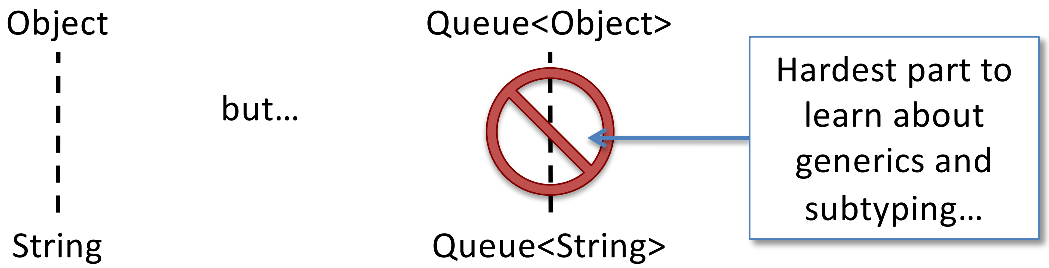
```
Queue<String> qs = new QueueImpl<>();
Queue<Object> qo = qs;

qo.enq(new Object());
String s = qs.deq();
```

Ok? Sure!
Ok? Let's see…

Ok? I guess
Ok? Noooo!

Java generics are *invariant*:

- Subtyping of *arguments* to generic types does not imply subtyping between instantiations:



Object

String

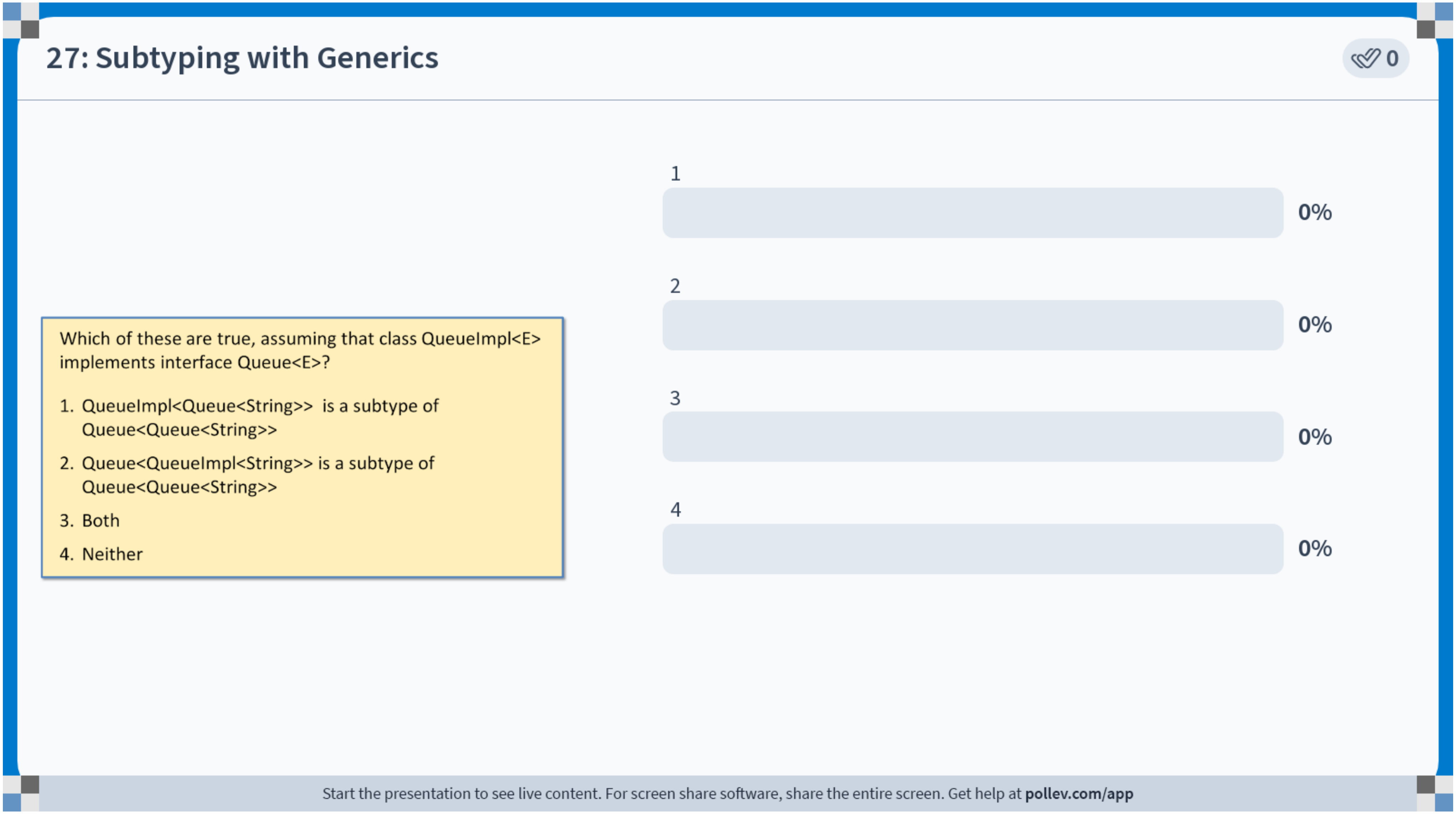
but…

Queue<Object>

Queue<String>

Hardest part to learn about generics and subtyping…

* Subtyping and generics interact in other ways too. Java supports *bounded polymorphism* and *wildcard types*, but those are beyond the scope of CIS 1200.



27: Subtyping with Generics

0

Which of these are true, assuming that class QueueImpl<E> implements interface Queue<E>?

1. QueueImpl<Queue<String>> is a subtype of Queue<Queue<String>>
2. Queue<QueueImpl<String>> is a subtype of Queue<Queue<String>>
3. Both
4. Neither

1

0%

2

0%

3

0%

4

0%

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Subtyping and Generics

Which of these are true, assuming that class QueueImpl<E> implements interface Queue<E>?

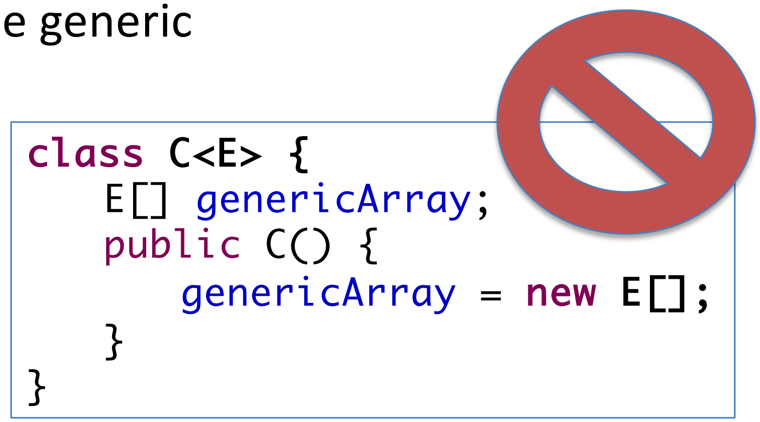
1. QueueImpl<Queue<String>> is a subtype of Queue<Queue<String>>
2. Queue<QueueImpl<String>> is a subtype of Queue<Queue<String>>
3. Both
4. Neither

Answer: 1

Other subtleties with Generics

- Unlike OCaml, Java classes and methods can be generic only with respect to *reference* types.
  - Not possible to do: `Queue<int>`
  - Must instead do: `Queue<Integer>`

- Java Arrays cannot be generic
  - Not possible:



```
class C<E> {
    E[] genericArray;
    public C() {
        genericArray = new E[];
    }
}
```