Programming Languages and Techniques (CIS120)

Lecture 26

November 2nd, 2015

Java ASM

Generics

Announcements

- *Midterm 2 is Friday, November 6th in class*
  - Last names A – L Leidy Labs 10 (here)
  - Last names M – Z Cohen G17
  - Everything starting with mutable state in Ocaml to Friday's

- Review Session:
  Wednesday, November 4th
  Levine 100
  7-9PM
  Pizza!

- My office hours today: 3:30 – 4:30

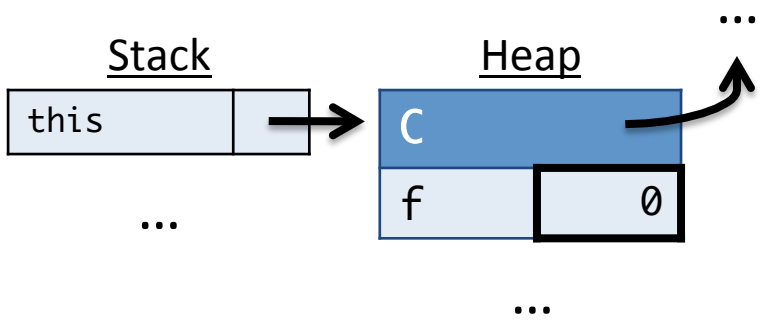
this

- Inside a non-static method, the variable `this` is a reference to the object on which the method was invoked.

- References to local fields and methods have an implicit "`this.`" in front of them.

```
class C {
    private int f;

    public void copyF(C other) {
        this.f = other.f;
    }
}
```



An 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();
```

Example

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

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

// … somewhere in main:
Decr d = new Decr(2);
d.dec();
int x = d.get();
```

How comfortable do you feel with the abstract stack machine?

1. Terrible, I'm totally lost.
2. A little shaky, but OK
3. Pretty comfortable
4. Great! Not a problem.

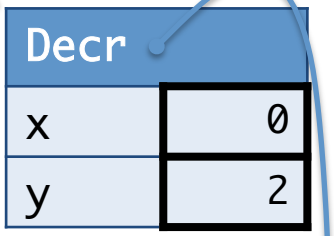
After the declaration of d

Workspace

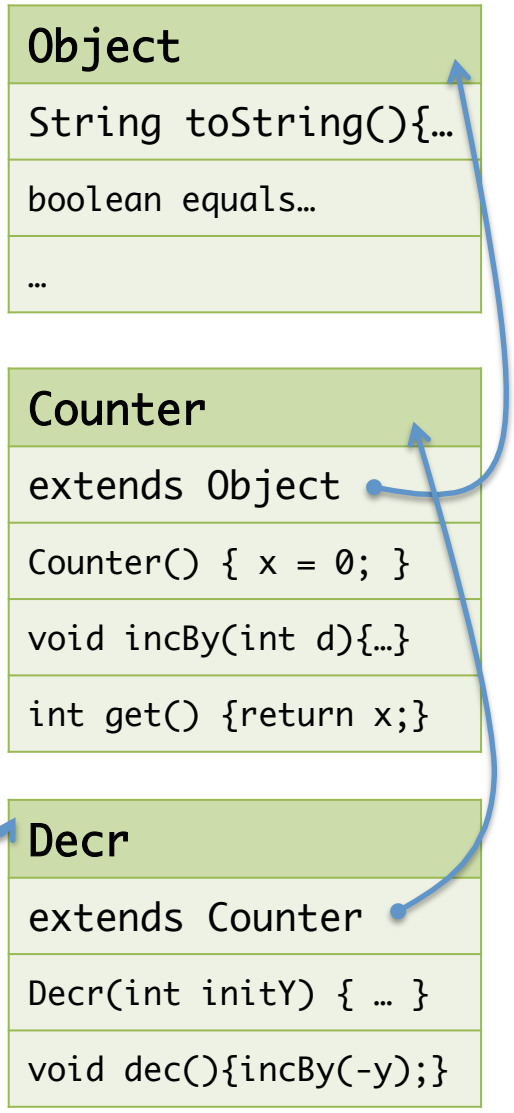
```
d.dec();
int x = d.get();
```

Stack

Heap



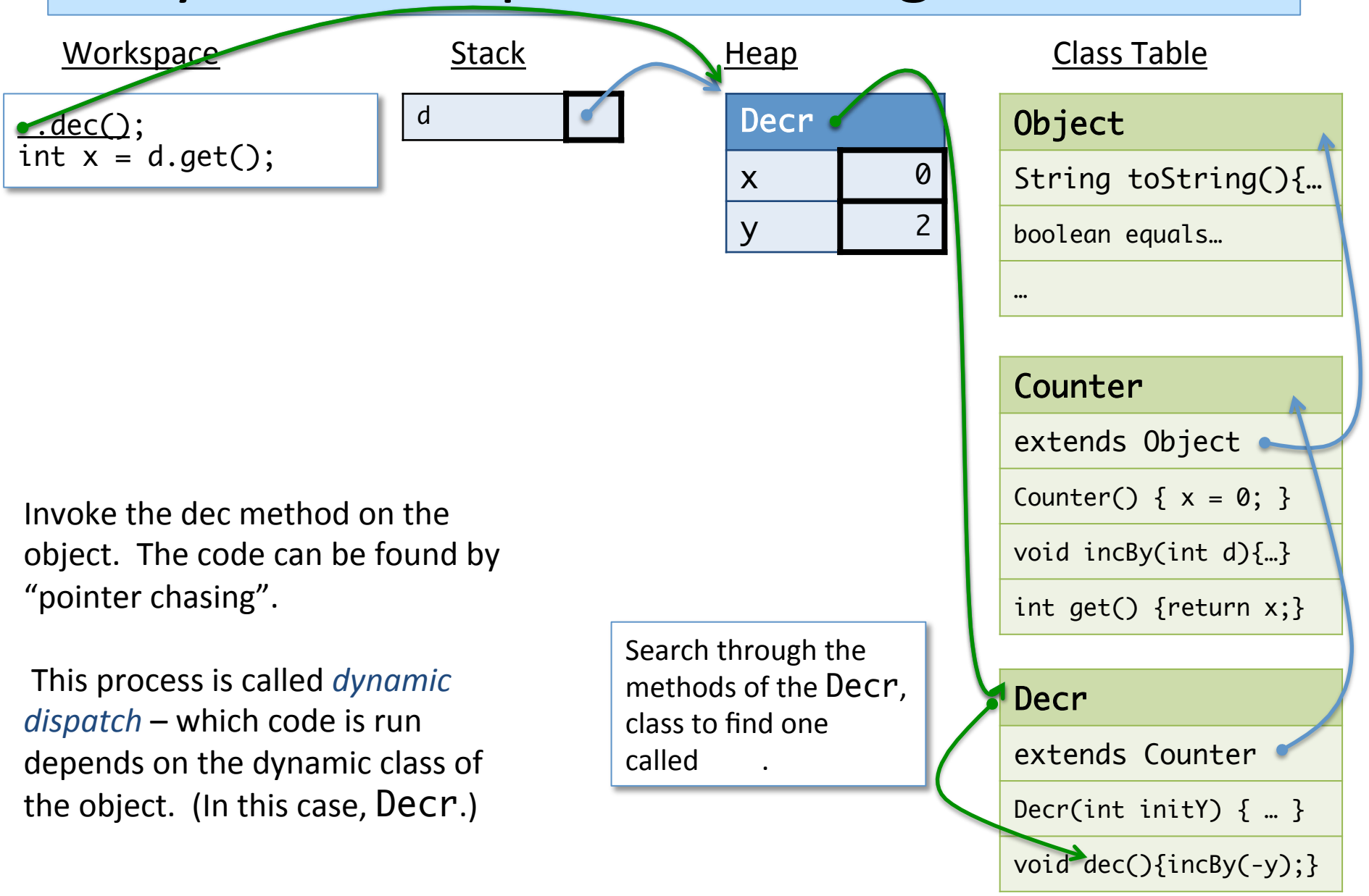
Class Table



Allocate a stack slot for the local variable d. It's mutable… (see the bold box in the diagram).

Aside: since, by default, fields and local variables are mutable, we often omit the bold boxes and just assume the contents can be modified.

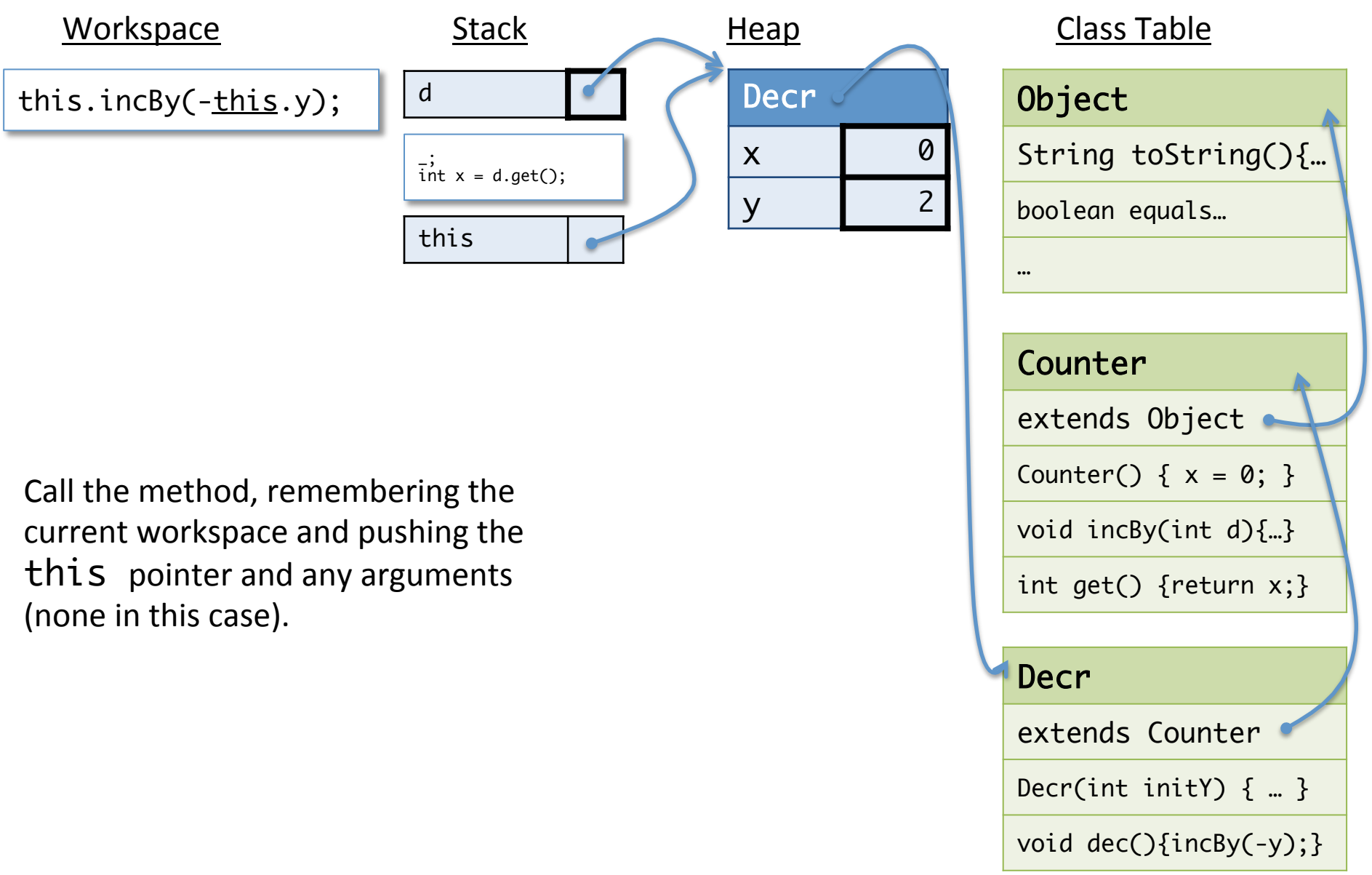
Dynamic Dispatch: Finding the Code



Invoke the dec method on the object. The code can be found by "pointer chasing".

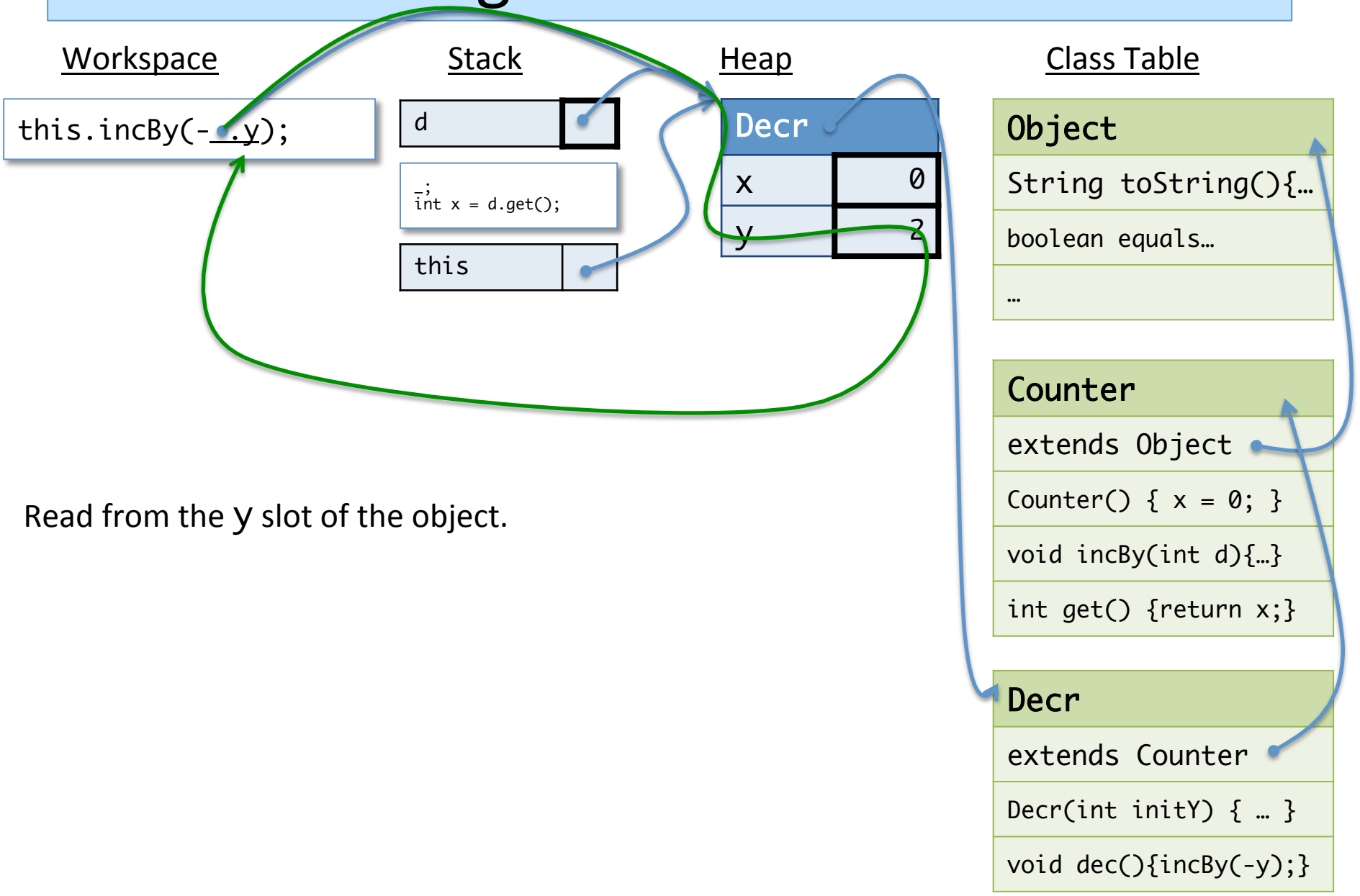
This process is called *dynamic dispatch* – which code is run depends on the dynamic class of the object. (In this case, Decr.)

Dynamic Dispatch: Finding the Code



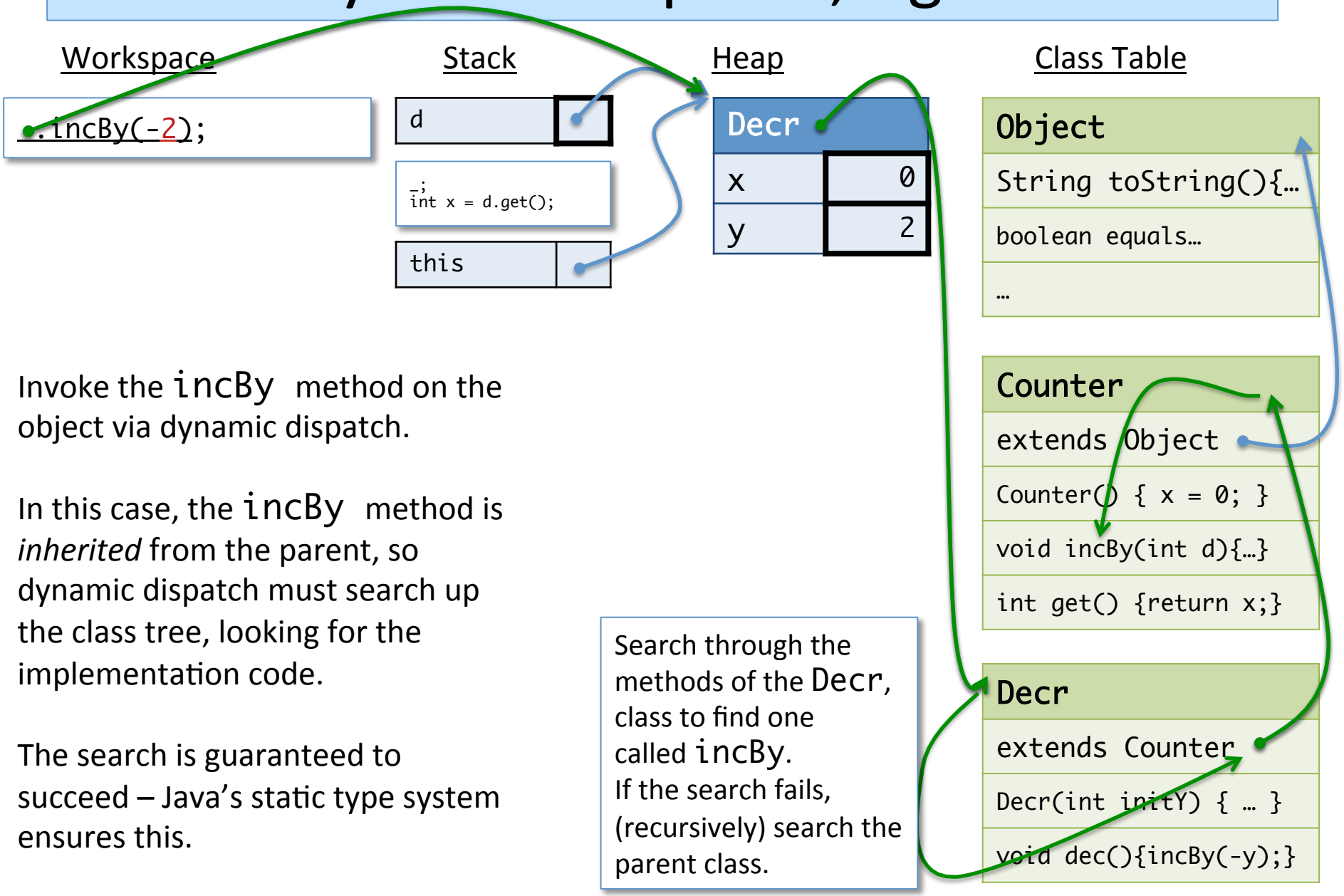
Call the method, remembering the current workspace and pushing the `this` pointer and any arguments (none in this case).

Reading A Field's Contents



Read from the y slot of the object.

Dynamic Dispatch, Again



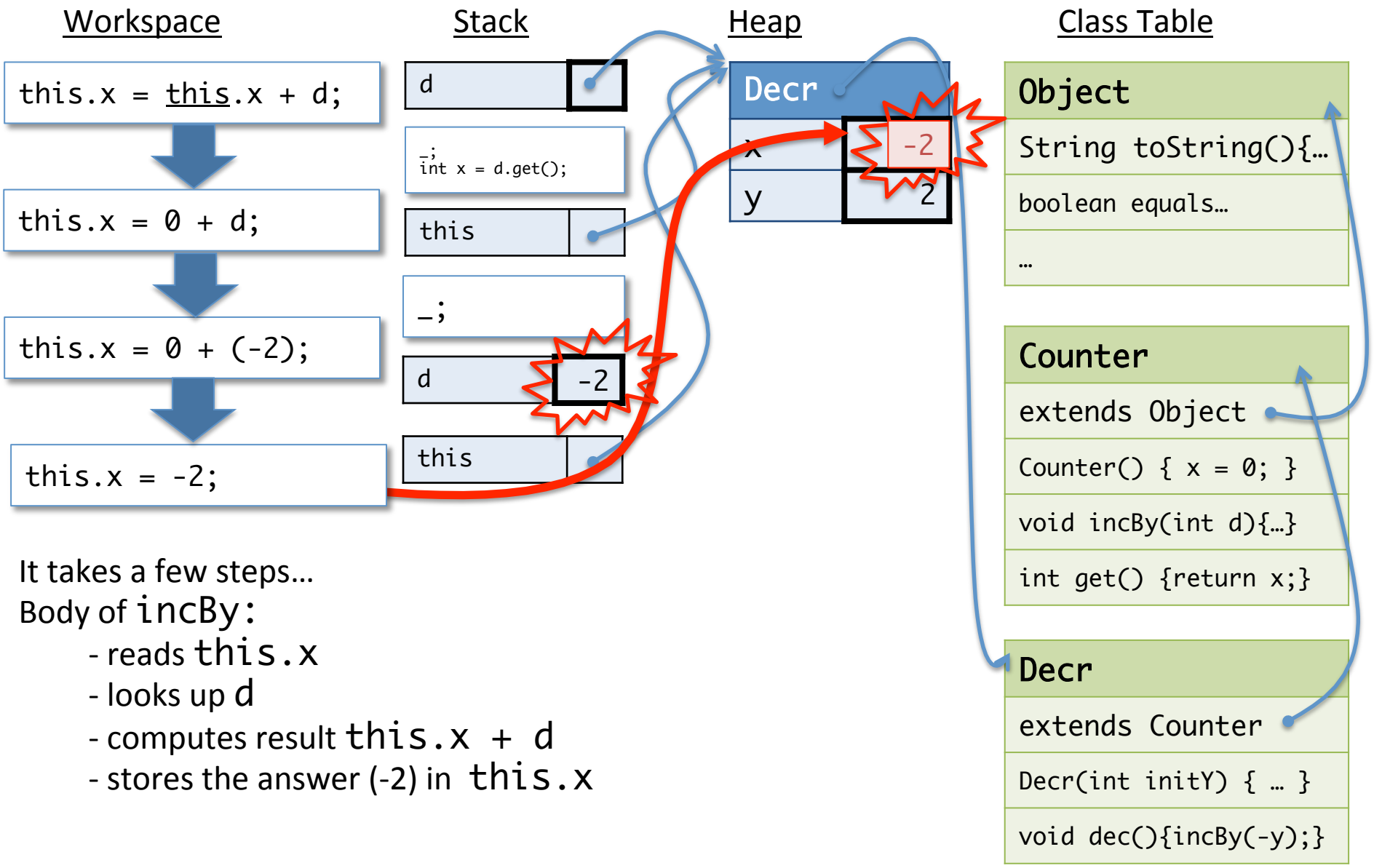
Invoke the `incBy` method on the object via dynamic dispatch.

In this case, the `incBy` method is *inherited* from the parent, so dynamic dispatch must search up the class tree, looking for the implementation code.

The search is guaranteed to succeed – Java's static type system ensures this.

Search through the methods of the `Decr`, class to find one called `incBy`.
If the search fails, (recursively) search the parent class.

Running the body of `incBy`

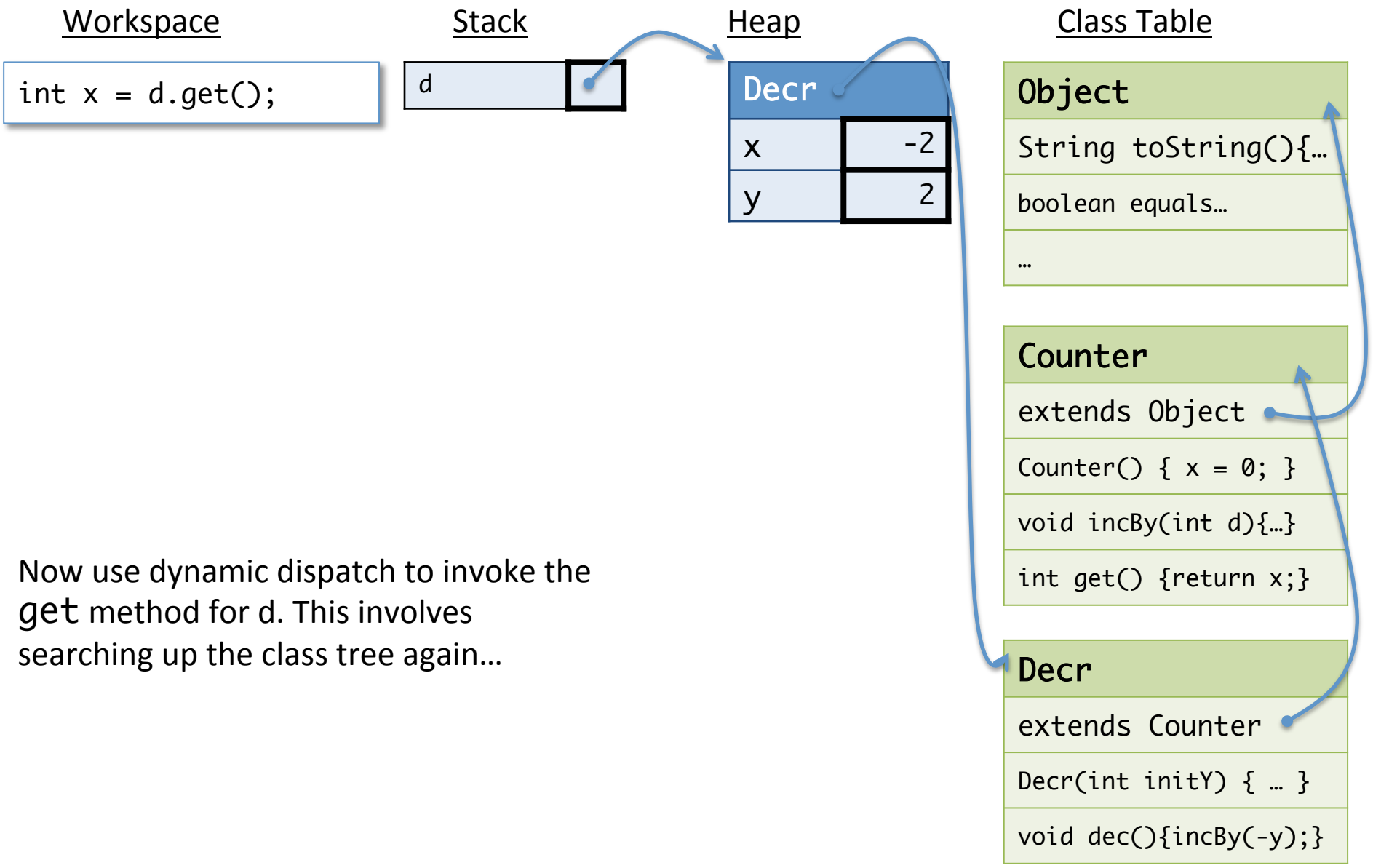


It takes a few steps…

Body of `incBy`:

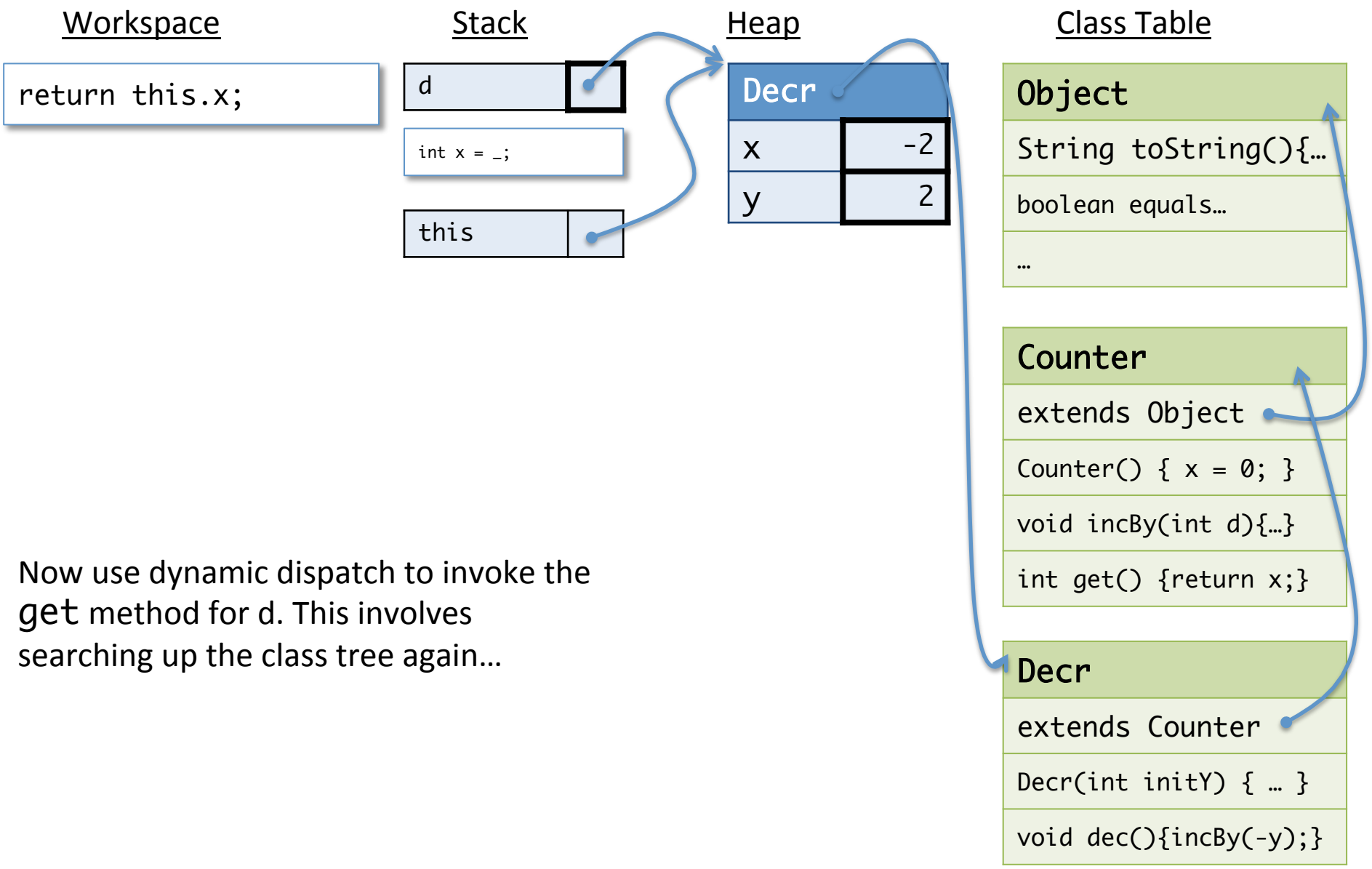
- reads `this.x`
- looks up `d`
- computes result `this.x + d`
- stores the answer (-2) in `this.x`

After a few more steps…



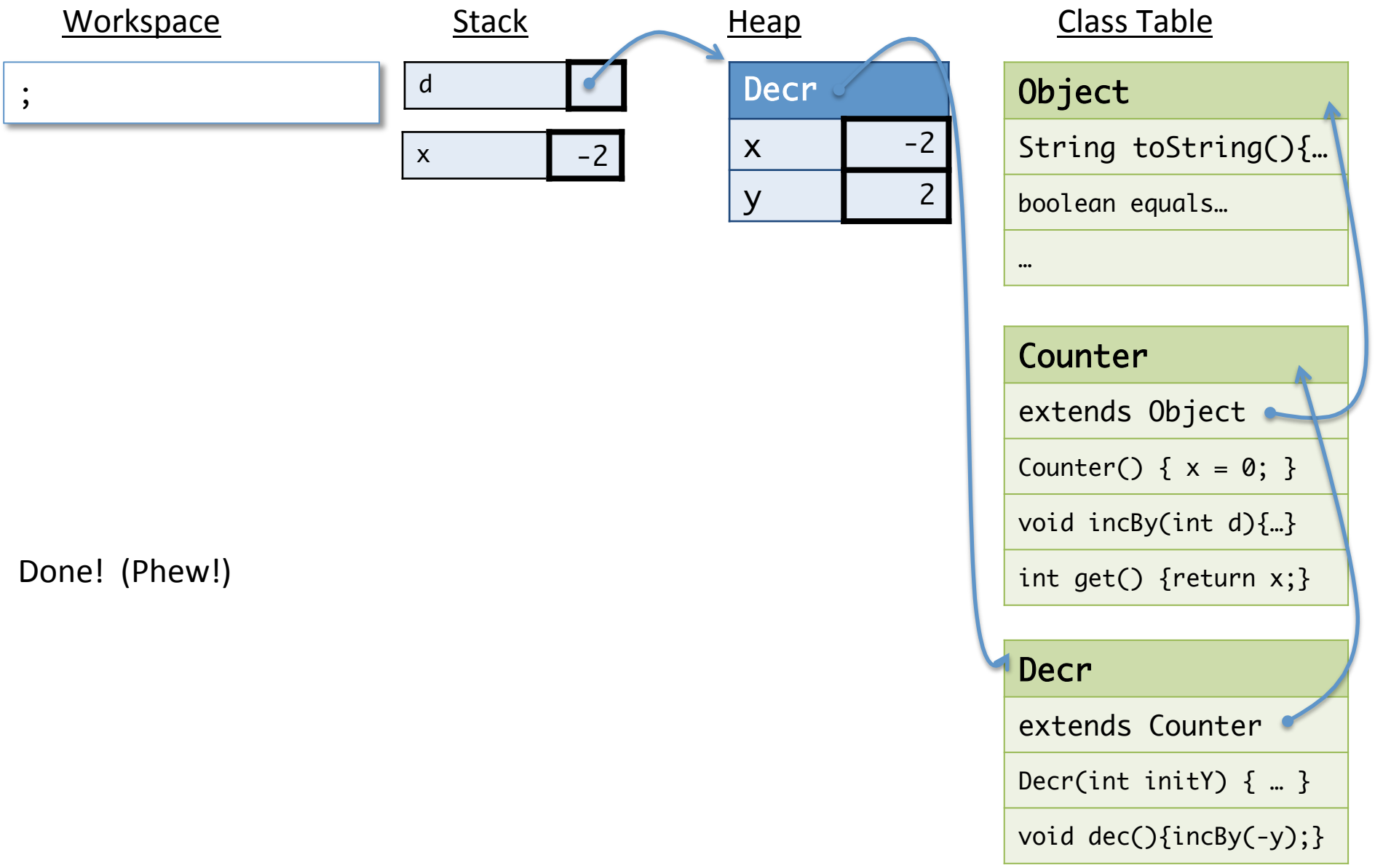
Now use dynamic dispatch to invoke the `get` method for d. This involves searching up the class tree again…

After a few more steps…



Now use dynamic dispatch to invoke the `get` method for d. This involves searching up the class tree again…

After yet a few more steps…



Done! (Phew!)

Summary: `this` and dynamic dispatch

- When object's method is invoked, as in `o.m()`, the code that runs is determined by `o`'s *dynamic* class.
  - The dynamic class, which is just a pointer to a class, is included in the object structure in the heap.
  - If the method is inherited from a superclass, determining the code for `m` might require searching up the class hierarchy via pointers in the class table.
  - This process is called *dynamic dispatch* (the heart of OOP!)

- Once the code for m has been determined, a binding for `this` is pushed onto the stack.
  - The `this` pointer is used to resolve field accesses and method invocations inside the code.

Static members & Java ASM

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

1. No
2. Yes
3. I’m not sure

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.

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

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

You can do a static assignment to initialize a static field.

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

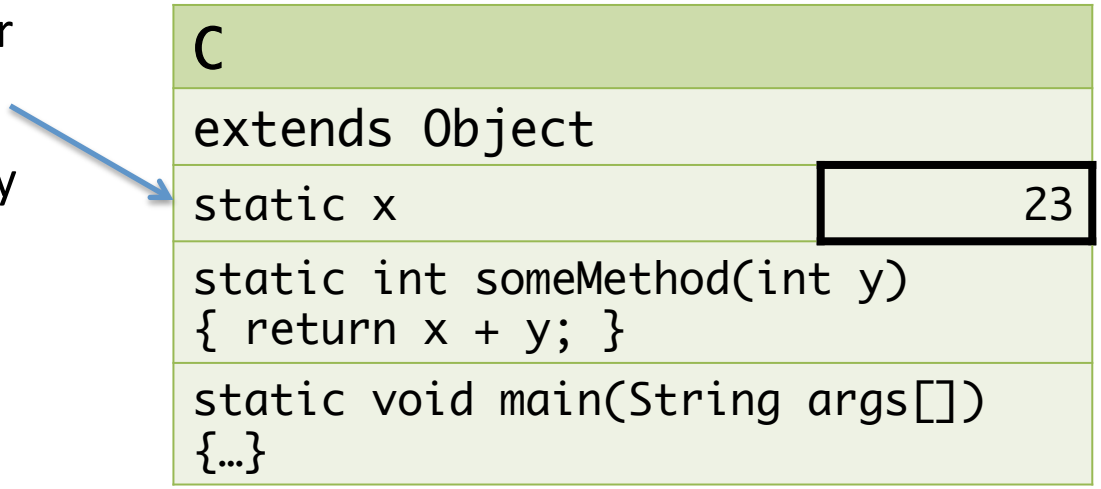
Example of Statics

- The java.lang.Math library provides static fields/methods for many common arithmetic operations:

- `Math.PI == 3.141592653589793`
- `Math.sin, Math.cos`
- `Math.sqrt`
- `Math.pow`
- etc.

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;`



- 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 globally visible (if the field is public)
  - Generally not a good idea!

Static Methods (Details)

- Static methods do *not* have access to the `this` pointer
  - 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
  - Eclipse will issue a warning if you try to do this.

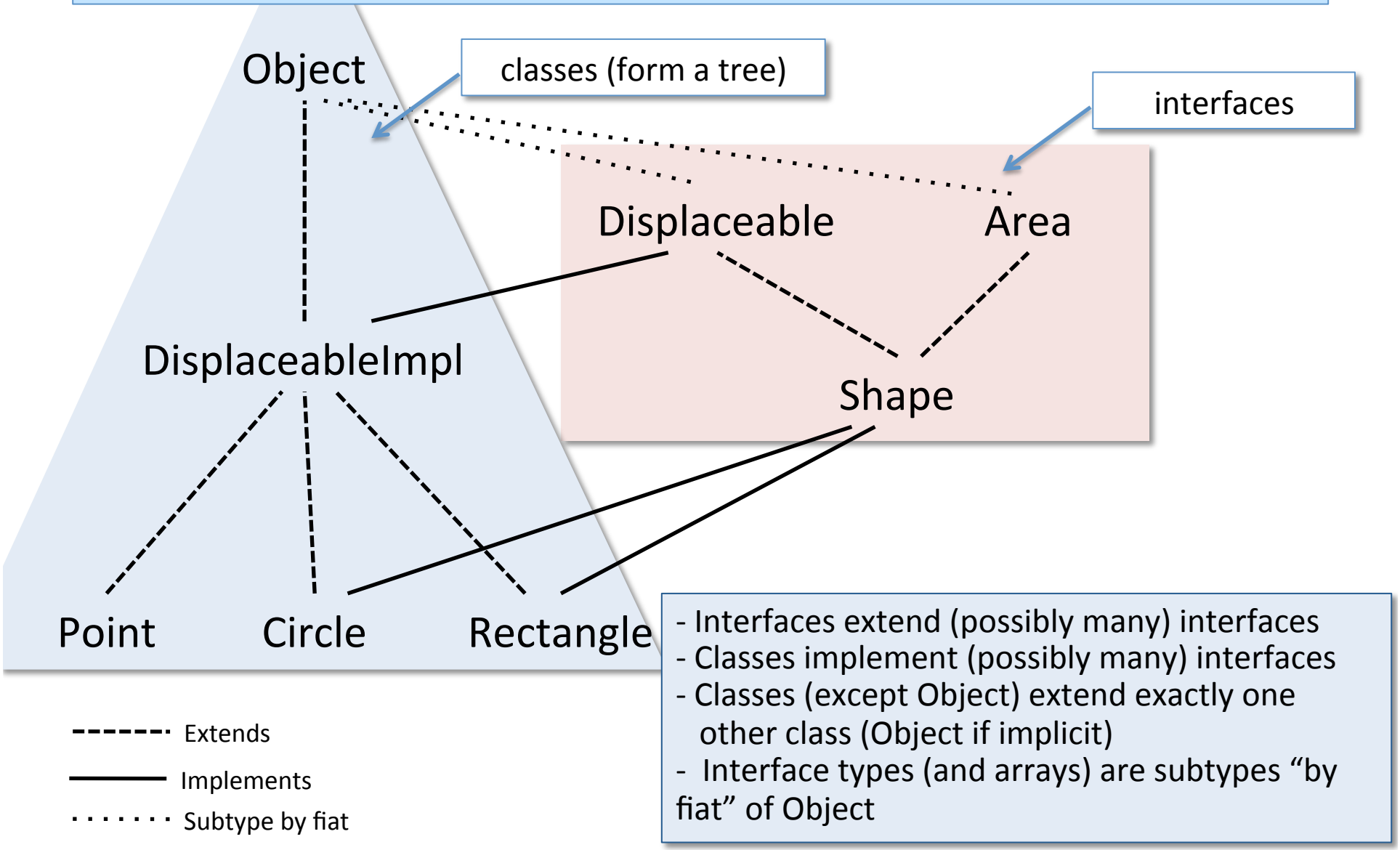
Java Generics

Object

```
public class Object {
  boolean equals(Object o) {
    … // test for equality
  }
  String toString() {
    … // return a string representation
  }
  …  // other methods omitted
}
```

- Object is the root of the class tree.
  - Classes that leave off the “extends” clause *implicitly* extend Object
  - Arrays also implement the methods of Object
  - This class provides methods useful for *all* objects to support
- Object is the highest type in the subtyping hierarchy.

Recap: Subtyping



- Interfaces extend (possibly many) interfaces
- Classes implement (possibly many) interfaces
- Classes (except Object) extend exactly one other class (Object if implicit)
- Interface types (and arrays) are subtypes "by fiat" of Object

Subtype Polymorphism*

- Main idea:

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

```
void method(A obj) {
     // use obj at type A
}

method(new B());
```

- If B is a subtype of A, it provides all of A's (public) methods.
- Due to dynamic dispatch, the behavior of the method depends on B's implementation.
  - Behavior of B should be "compatible" with A's behavior
  - Simple inheritance makes this easier

*polymorphism = many shapes

Is subtyping good enough?

Subtype Polymorphism

vs.

Parametric Polymorphism

Mutable Queue ML Interface

```
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

  (* Remove the first occurrence of the value. *)
  val remove : 'a -> 'a queue -> unit
end
```

Subtype Polymorphism

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

  …
}
```

```
ObjQueue q = …;

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

What type for A?

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