ture #9: Interfaces and Abstract Classes

r projects are individual efforts in this class (no). Feel free to discuss projects or pieces of them the work. But you must complete and write up each self. That is, feel free to discuss projects with each e aware that we expect your work to be substantially om that of all your classmates (in this or any other

Public Service Announcement

one! We are calling all hackers, makers, and 4am'ers 4.0—the worlds largest collegiate hackathon held ley! Registration is now open to UC Berkeleys. Get night full of hacking, making awesome programming rning new APIs, meeting CEOs and tech executives, food, and so much learning!
now at www.calhacks.io (and we recommend you apply

Recreation

ny polynomial with a leading coefficient of 1 and integral rational roots are integers.

Methods on Drawables

```
drawable object. */
 abstract class Drawable {
Expand THIS by a factor of SIZE */
ic abstract void scale(double size);
Draw THIS on the standard output. */
ic abstract void draw();
```

ew Drawable(), *BUT*, we can write methods that operate s in Drawable or in other classes:

```
l(Drawable[] thingsToDraw) {
wable thing : thingsToDraw)
draw();
```

no implementation! How can this work?

Abstract Methods and Classes

thod can be *abstract:* No body given; must be supplied

e is in specifying a pure interface to a family of types:

```
ble object. */
ract class Drawable {
act class" = "can't say new Drawable"
d THIS by a factor of SIZE */
stract void scale(double size);
THIS on the standard output. */
stract void draw();
```

ble is something that has *at least* the operations scale it.

a Drawable because it's abstract.

his case, it wouldn't make any sense to create one, be- two methods without any implementation.

Using Concrete Classes

te new Rectangles and Circles.

classes are subtypes of Drawable, we can put them in r whose static type is Drawable, ...

fore can pass them to any method that expects Drawable

```
e[] things = {
 Rectangle(3, 4), new Circle(2)

(things);
```

l rectangle and a circle with radius 2.

Concrete Subclasses

ses can extend abstract ones to make them "less ab- verriding their abstract methods.

inds of Drawables that are *concrete*, in that all methods entations and one can use **new** on them:

```
Rectangle extends Drawable {
angle(double w, double h) { this.w = w; this.h = h; }
 scale(double size) { w *= size; h *= size; }
 draw() { draw a w x h rectangle }
ble w,h;
```

r Circle or Rectangle is a Drawable.

```
Circle extends Drawable {
le(double rad) { this.rad = rad; }
 scale(double size) { rad *= size; }
 draw() { draw a circle with radius rad }
ble rad;
```

Interfaces

nglish usage, an *interface* is a "point where interaction een two systems, processes, subjects, etc." (*Concise ionary*).

ning, often use the term to mean a *description* of this raction, specifically, a description of the functions or which two things interact.

e term to refer to a slight variant of an abstract class ava 1.7) contains only abstract methods (and static con- this:

```
ace Drawable {
double size);   // Automatically public.
;
```

re automatically abstract: can't say `new Drawable();` Rectangle(...).

Implementing Interfaces

eat Java interfaces as the public specifications of data asses as their implementations:

```
s Rectangle implements Drawable { ... }
```

nterface as for abstract classes:

```
l(Drawable[] thingsToDraw) {
wable thing : thingsToDraw)
draw();
```

orks for Rectangles and any other implementation of

Multiple Inheritance

ne class, but *implement* any number of interfaces.

ample:

```
dable {
);

table {
ject x);

implements Readable {
ct get() { ... }
```

```
void copy(Readable r,
          Writable w) {
  w.put(r.get());
}

class Sink implements Writable {
  public void put(Object x) { ... }
}
```

```
class Variable implements Readable, Writable {
  public Object get() { ... }
  public void put(Object x) { ... }
}
```

gument of `copy` can be a `Source` or a `Variable`. The e a `Sink` or a `Variable`.

Review: Higher-Order Functions

u had *higher-order functions* like this:

```
c,      items):
ion      list
 is None:
rn None

rn IntList(proc(items.head), map(proc, items.tail))
```

d write

```
, makeList(-10, 2, -11, 17))
 makeList(10, 2, 11, 17)
bda x: x * x, makeList(1, 2, 3, 4))
 makeList(t(1, 4, 9, 16)
```

t have these directly, but can use abstract classes or nd subtyping to get the same effect (with more writing)

Map in Java

```
n one integer argument */

 IntUnaryFunction {
k);
```

```
IntList map(IntUnaryFunction proc,
            IntList items) {
  if (items == null)
    return null;
  else return new IntList(
      proc.apply(items.head),
      map(proc, items.tail)
    );
}
```

of this function that's clumsy. First, define class for e function; then create an instance:

```
mplements IntUnaryFunction {
t apply(int x) { return Math.abs(x); }

-----------------------------------
 Abs(), some list);
```


Lambda Expressions

e can create classes likes `Abs` on the fly with *anonymous*

```
 IntUnaryFunction() {
public int apply(int x) { return Math.abs(x); }
some list);
```

f like declaring

```
mous implements IntUnaryFunction {
int apply(int x) { return Math.abs(x); }
```

ting

```
 Anonymous(), some list);
```

ful (albeit Dangerous) Features of Java 8

above, before Java 8, interfaces contained just static
d abstract methods.

lement multiple interfaces, but extend only one class:
rface inheritance, but *single body inheritance*.

is simple, and pretty easy for language implementors to

ere are cases where it would be nice to be able to "mix
tations from a number of sources.

duced static methods into interfaces and also *default*
ich are essentially instance methods and are used when-
d of a class implementing the interface would otherwise

re, but, as in other languages with full multiple inheri-
++ and Python), it can lead to confusing programs.

that the new default method feature should be used

Lambda in Java 8

mbda expressions are even more succinct:

```
t x) -> Math.abs(x), some list);
```

better, when the function already exists:

```
h::abs, some list);
```

out you need an anonymous `IntUnaryFunction` and cre-

examples in `game2048.GUI`:

```
Button("Game->New", this::newGame);
```

cond parameter of `ucb.gui2.TopLevel.addMenuButton`
function.

ava library type `java.util.function.Consumer`, which
gument method, like `IntUnaryFunction`,