LEC 14

CSE 122

Interfaces

BEFORE WE START

Talk to your neighbors:

What is your most-used emoji?

Instructors Tristan Huber & Hunter Schafer

TAs

Ambika	Evelyn	Poojitha
Andrew	Jacob	Rishi
Audrey	Jaylyn	Rucha
Autumn	Jin	Shivani
Ayush	Joe	Shreya
Ben	Kevin	Steven
Colton	Leon	Suhani
Di	Megana	Yijia
Eesha	Melissa	Ziao
Elizabeth	Mia	

Questions during Class?

Raise hand or send here

sli.do #cse122

Lecture Outline

- **Announcements** ◀
- Interfaces Review
- More Shapes!
- Comparable

Announcements

- C2 due tomorrow (Thurs, May 18)
- P3 will be released on Fri, May 19
- Quiz 2 next Tuesday (May 23)
- Reminder that the final exam is scheduled for **Tuesday (June 6) 2:30pm-4:20pm**

Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- Comparable

(PCM) Interfaces

Interfaces - define a set of *behavior* which classes can implement

… like a "certification", eg `ArrayList` is "certified" as a `List` because it can do all `List` behaviors.

note: interfaces say nothing about a class' *state*

(PCM) `List` Interface

`List` is an interface – defines the *behaviors* which make something a List, inc:

`add`, `clear`, `contains`, `get`, `isEmpty`, `size`

Any class with these behaviors can implement `List`

List documentation enumerates the full list of methods required to be a List:
https://docs.oracle.com/javase/8/docs/api/java/util/List.html

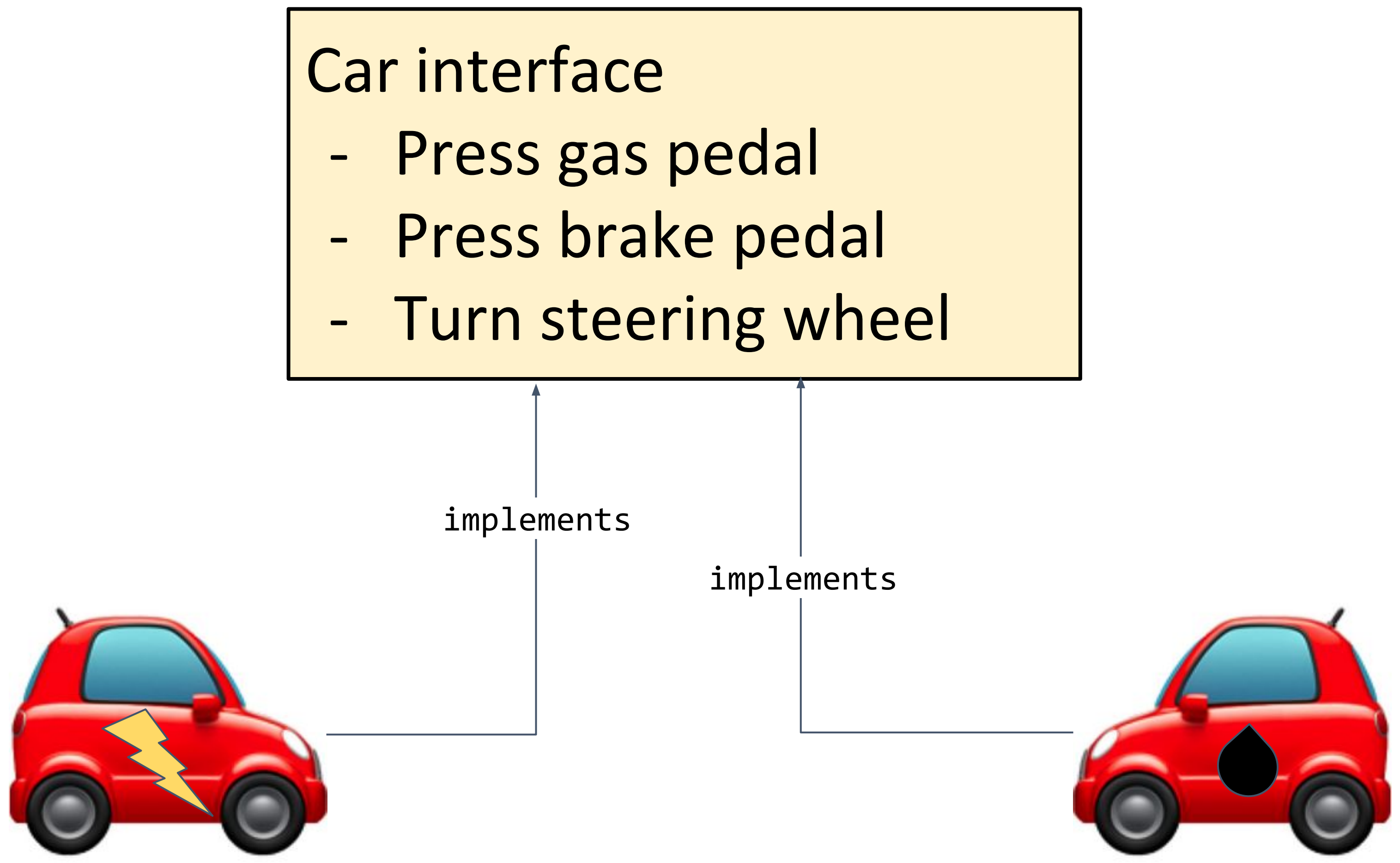
(PCM) Why interfaces?

Abstraction

Interfaces support *abstraction*

(the separation of ideas from details)

(PCM) Why interfaces?


Car interface
- Press gas pedal
- Press brake pedal
- Turn steering wheel

(PCM) Why interfaces?

Flexibility

```
public static void driveToWork(Car c) {…}
```

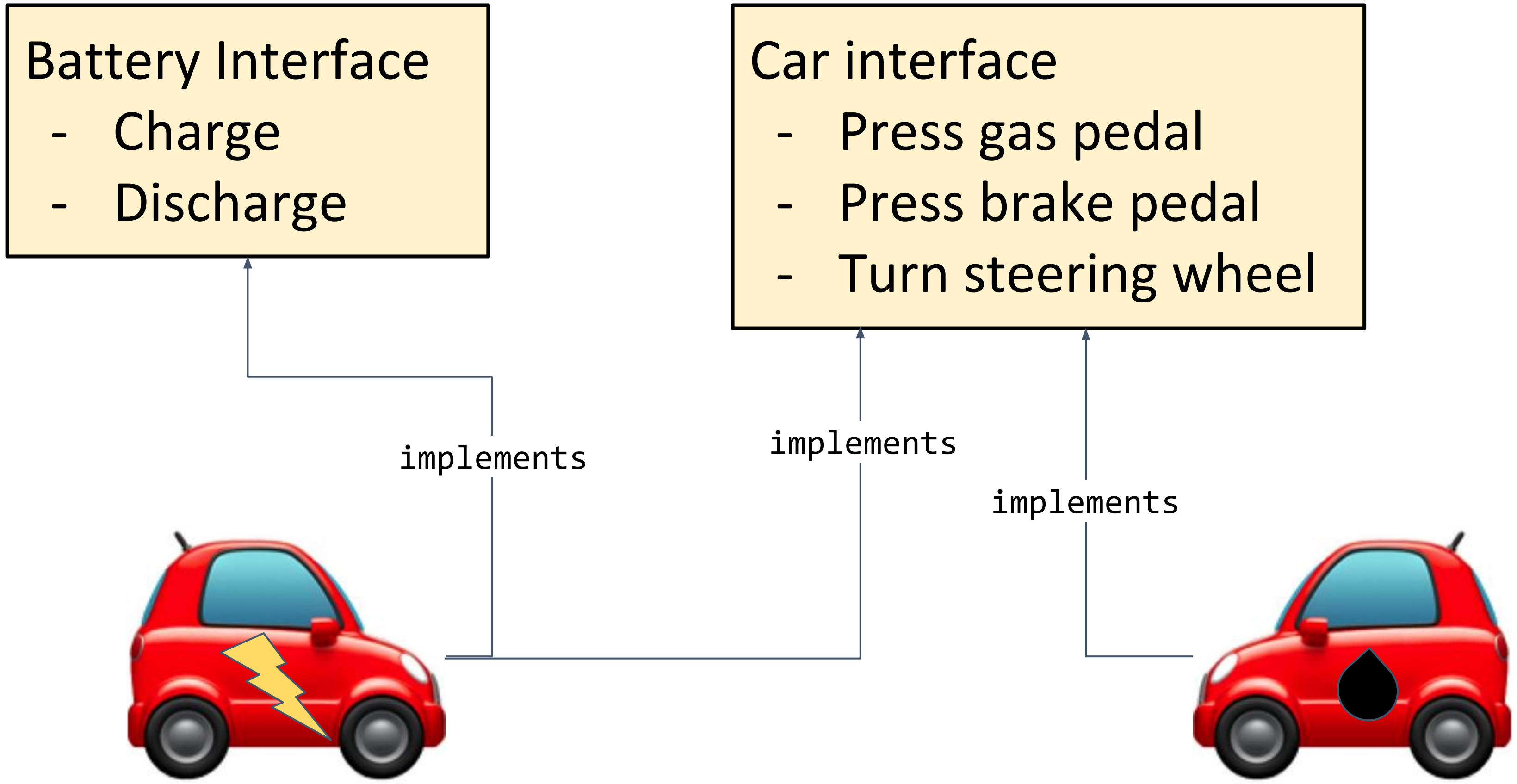
This method does not need to change if we update our implementation of `Car`

Lecture Outline

- Announcements
- Interfaces Review
- **More Shapes!** ◀
- Comparable

Classes can Implement Multiple Interfaces

A class can implement multiple interfaces – must include all *behaviors* from each interface it implements



Classes can Implement Multiple Interfaces

```
public class Leaf implements Car, Battery {
    ...
}
```

But `Leaf` would have to implement:

- `pushGasPedal`, etc from `Car`

  AND

- `charge`, `discharge` from `Battery`

An interface can extend another

You can have one interface *extend* another

So if `public interface A extends B`, then to implement A a class needs to have all methods from A and B.

In the above example, A is *more specific* than B

An interface can extend another

We can write another interface

`Polygon` that extends `Shape`

- `Square` is a `Polygon` (and `Shape`)
- `Triangle` is a `Polygon` (and `Shape`)
- `Circle` is a `Shape` (but *not* a `Polygon`)

Practice : Think

sli.do #cse122

Select all of the following statements that would cause an error.

```
public interface A {
    public void a();
}

public interface B extends A {
    public void b();
}

public interface C {
    public void c();
}

public interface D extends A {
    public void d();

    public void e();
}
```

```
public class One implements A {
    ...
}

public class Two implements B, D {
    ...
}

public class Three implements B, C {
    ...
}
```

A)
```
B foo = new Two();
foo.b();
```

B)
```
D bar = new Two();
bar.a();
```

C)
```
D baz = new Three();
baz.a();
```

D)
```
A waldo = new Three();
waldo.b();
```

Practice : Pair

sli.do #cse122

Select all of the following statements that would cause an error.

```
public interface A {
    public void a();
}

public interface B extends A {
    public void b();
}

public interface C {
    public void c();
}

public interface D extends A {
    public void d();

    public void e();
}
```

```
public class One implements A {
    ...
}

public class Two implements B, D {
    ...
}

public class Three implements B, C {
    ...
}
```

A)
```
B foo = new Two();
foo.b();
```

B)
```
D bar = new Two();
bar.a();
```

C)
```
D baz = new Three();
baz.a();
```

D)
```
A waldo = new Three();
waldo.b();
```

Select all of the following statements that would cause an error.

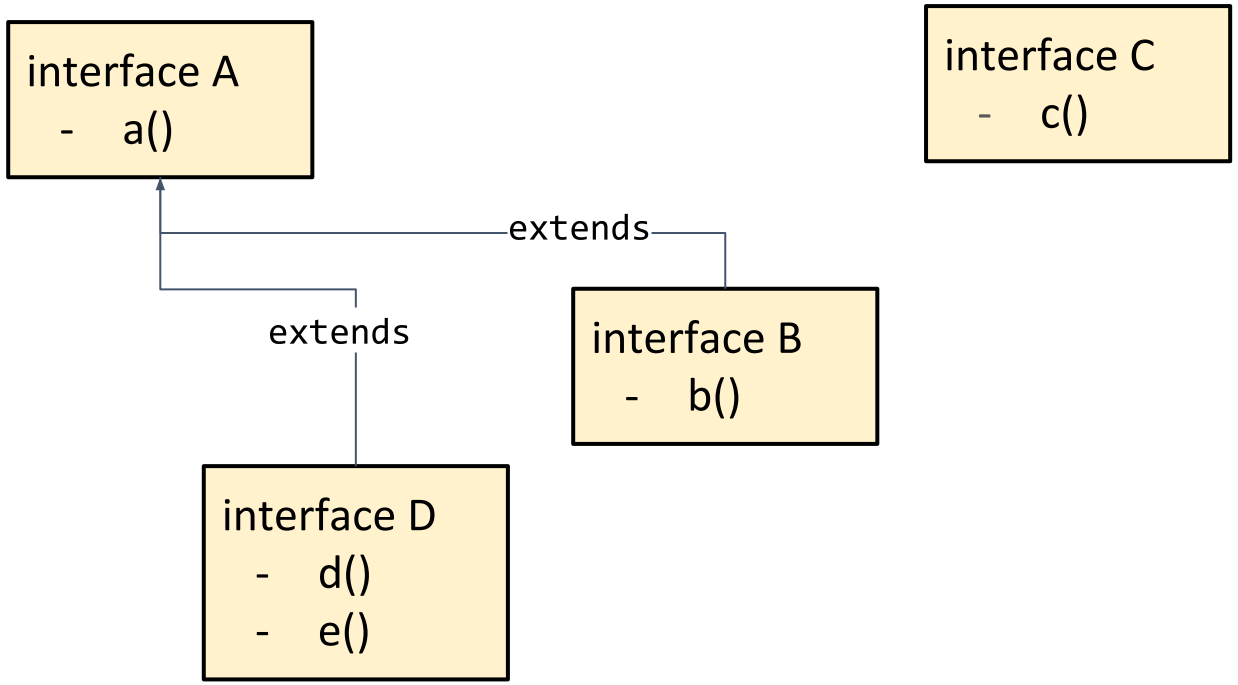
```
public interface A {
    public void a();
}

public interface B extends A {
    public void b();
}

public interface C {
    public void c();
}

public interface D extends A {
    public void d();

    public void e();
}
```

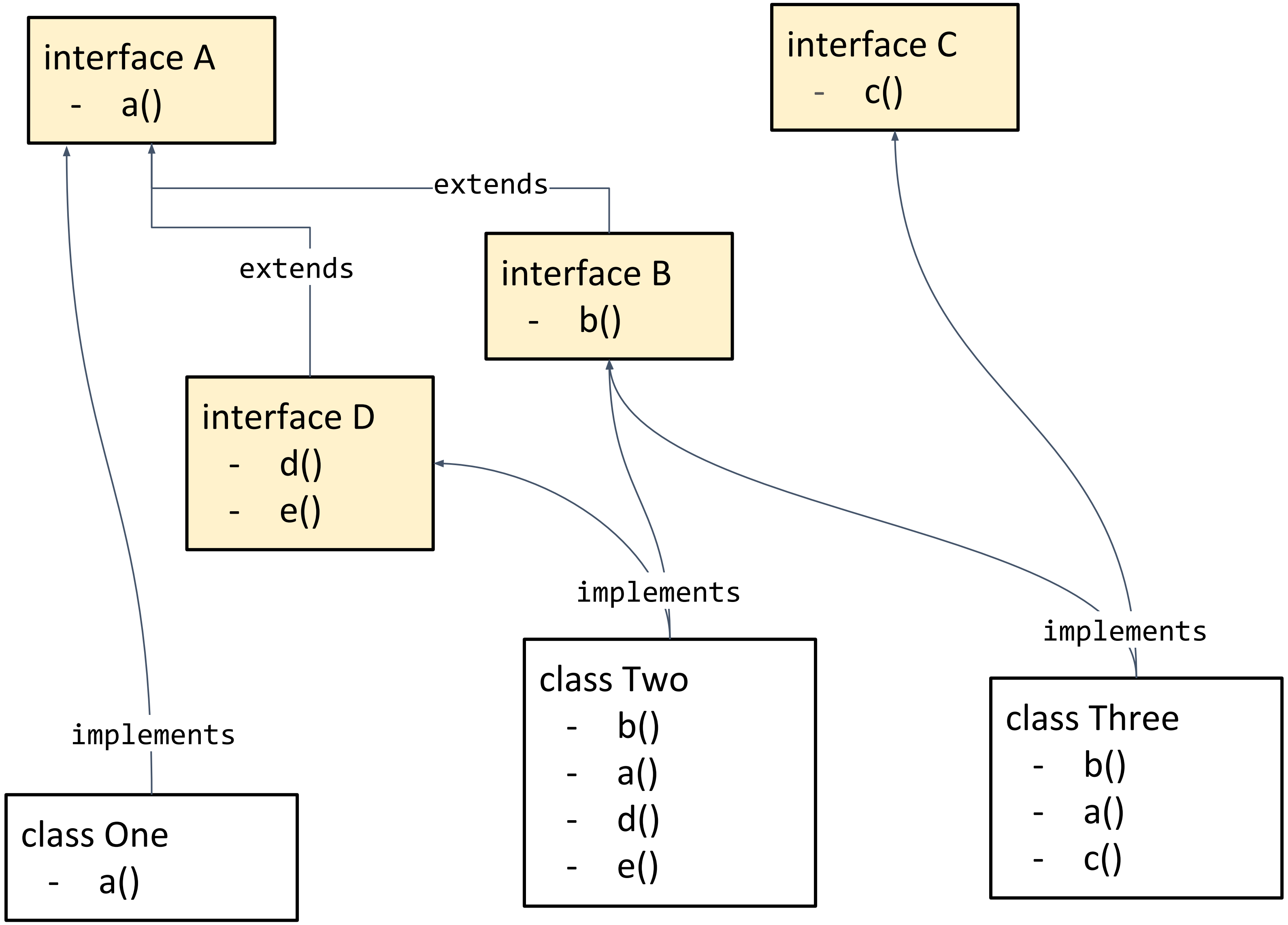


Select all of the following statements that would cause an error.

```
public class One implements A {
    ...
}

public class Two implements B, D {
    ...
}

public class Three implements B, C {
    ...
}
```



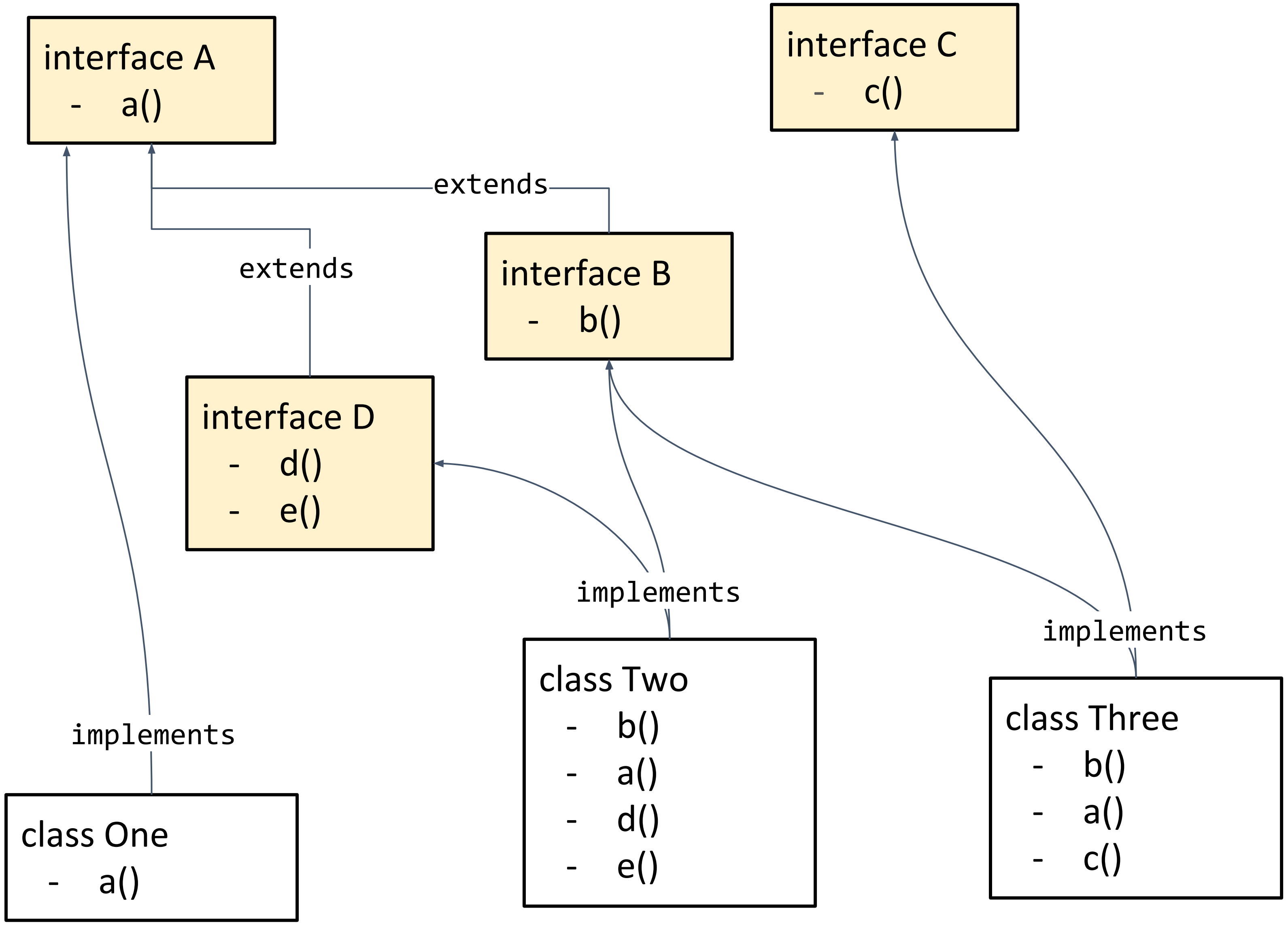
Select all of the following statements that would cause an error.

A)
```
B foo = new Two();
foo.b();
```

B)
```
D bar = new Two();
bar.a();
```

C)
```
D baz = new Three();
baz.a();
```

D)
```
A waldo = new Three();
waldo.b();
```



Lecture Outline

- Announcements
- Interfaces Review
- More Shapes!
- **Comparable**

Recall the Student / Course Example from Wed

Course stored a field

```
    private List<Student> roster;
```

We also had a suggestion to use a `Set` to store the students…

Seems like a great idea (no duplicates, not worried about keeping a specific order or indexing into it) but … Java reasons

- `HashSet` won't work because of the `hashCode()` business I mentioned on Wed
- `TreeSet` won't work because what does it mean to "sort" `Students`

Comparable

`TreeSet` uses an *interface* called `Comparable<E>` to know how to sort its elements

Only has one required method:

```
    public int compareTo(E other)
```

Its return value is:

```
< 0  if this is "less than" other
  0  if this is equal to other
> 0  if this is "greater than" other
```

`Comparable documentation:`
https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html