

LEC 02

CSE 123

Polymorphism; Abstract Classes

Questions during Class?
Raise hand or send here

sli.do #cse123



BEFORE WE START

Talk to your neighbors:

Coffee or tea? Or something else?

[Respond on sli.do!](#)

Instructor: Nathan Brunelle

TAs:

Cora	Chris	Crystal	Anya	Evan
Jonah	Nhan	Saachi	Ishita	Sahana
Chloe	Shiven	Nate	Renee	Yuntong
Amiya	Shashmit	Akshaya	Vihan	Shreedhaarini
Roshan	Adiyan	Nancy	Rachel	Ananya
Ryuta	Mimi	Isaac	Megan	

Announcements

- Programming Assignment 0 will be released tomorrow, Thursday, Oct 8 and is due Wednesday, Oct 21
- Computer Quiz 0 can be taken from Thursday, Oct. 22 through Thursday Oct. 29
 - Sign ups begin next week
 - A practice quiz will be released next week

Lecture Outline: Polymorphism

- Polymorphism 
- Compiler vs. Runtime Errors
- Abstract Classes
- Academic Honesty Policy

Declared Type and Actual Type

```
DeclaredType varName = new ActualType(...);
```

ActualType must be a subclass of (or same as) DeclaredType

Example 1

```
Animal rufus = new Dog("Rufus");
```

Declared Type: Animal

Actual Type: Dog

Can call methods that makes sense for EVERY Animal

If Dog overrides a method, uses the Dog version

Example 2

```
Dog rufus = new Dog("Rufus");
```

Declared Type: Dog

Actual Type: Dog

Can call methods that makes sense for EVERY Dog

If Dog overrides a method, uses the Dog version

Lecture Outline: Compile vs. Runtime Errors

- Polymorphism
- **Compiler vs. Runtime Errors** 
- Abstract Classes
- Academic Honesty Policy

Compiler vs. Runtime Errors

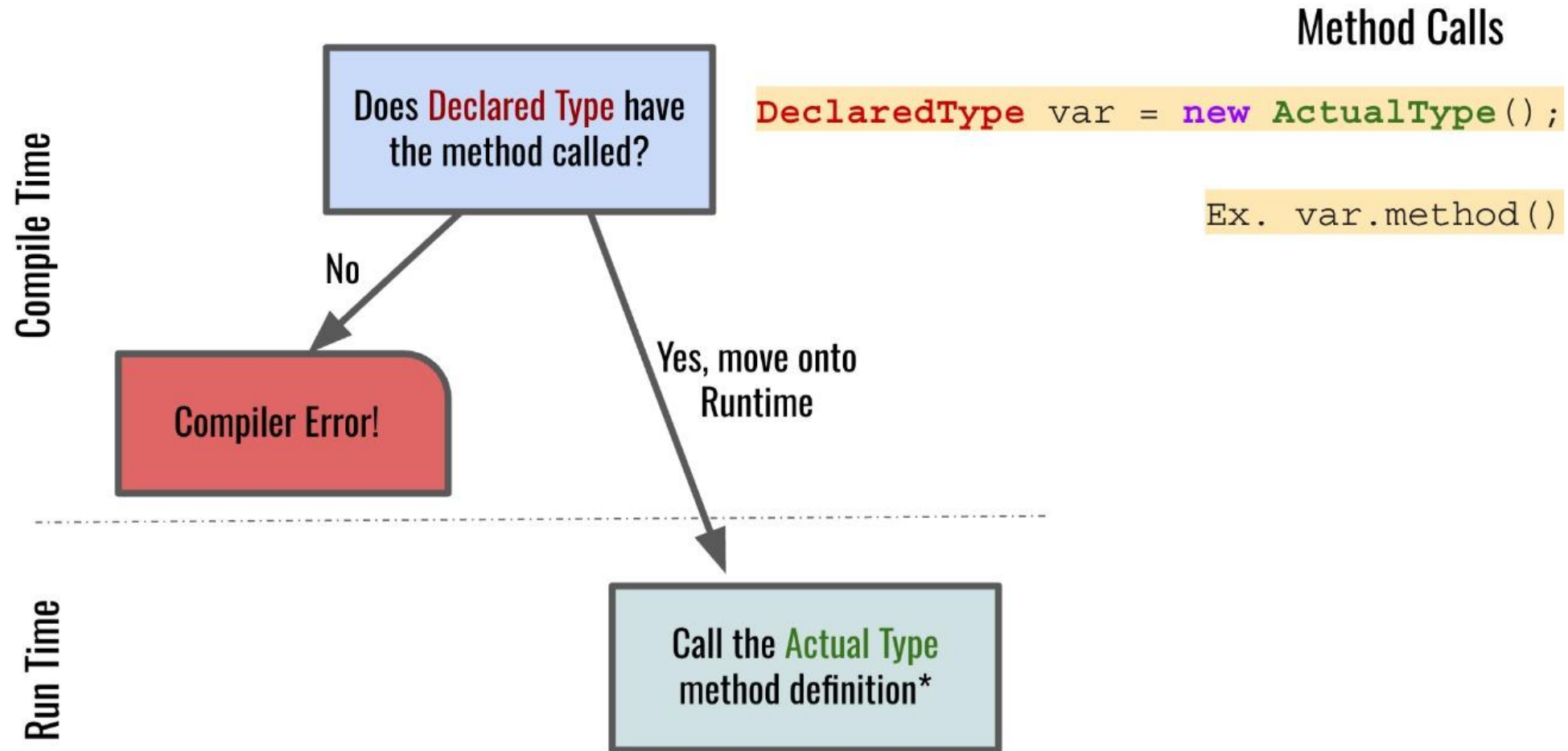
- **DeclaredType** x = new **ActualType**()
 - At compile time, Java only knows DeclaredType
 - Compiler error: trying to call a method that isn't present

```
Animal a = new Dog();  
a.bark(); // No bark() -> CE
```
 - Can cast to change the **DeclaredType** of an object

```
Dog d = (Dog) a;  
d.bark(); // No more CE
```
 - Runtime error: attempting cast to type that is not a superclass of actual type

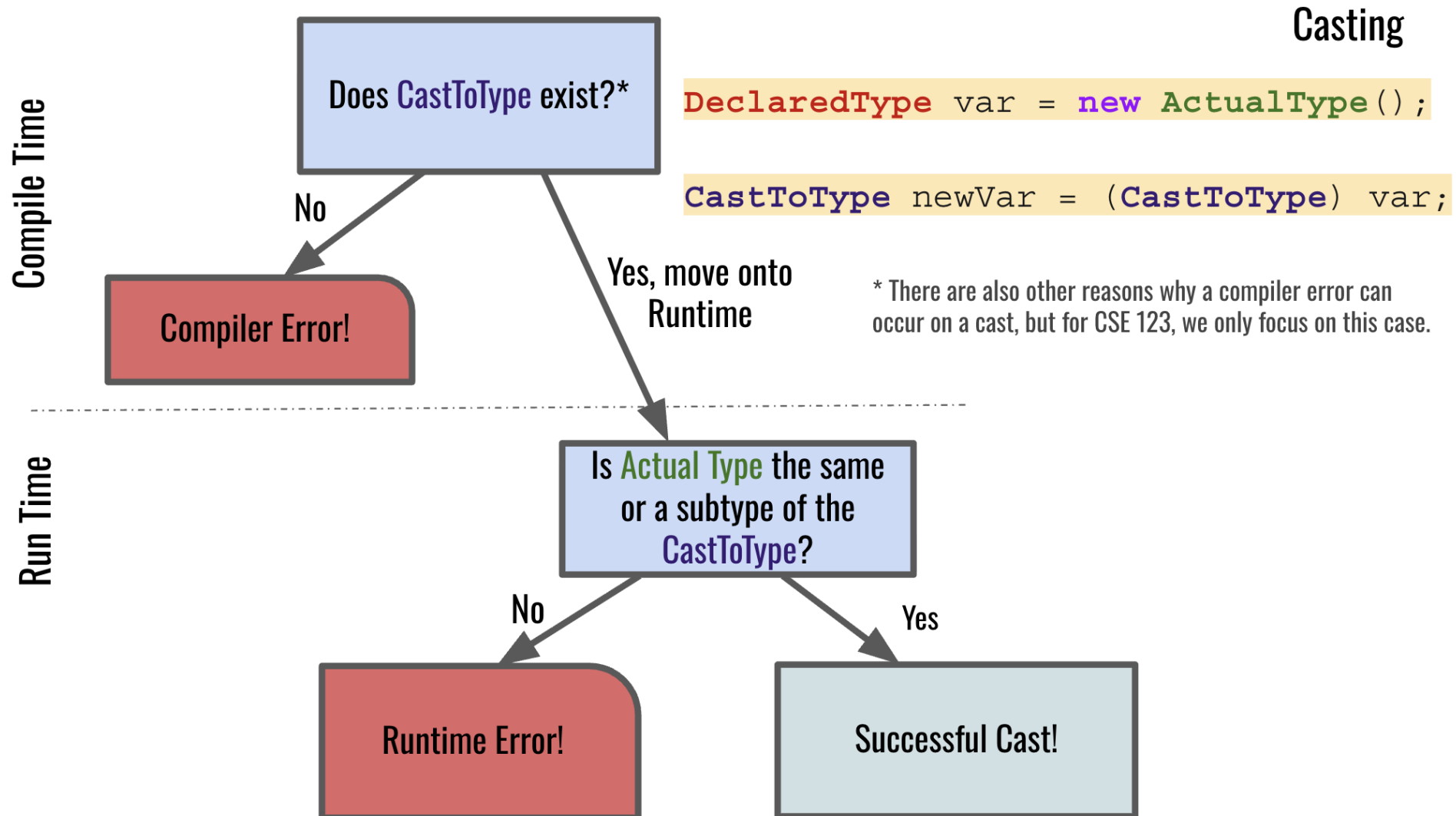
```
Animal a = new Fish();  
Dog d = (Dog) a; // Can't cast -> RE  
d.bark();
```
 - Order matters! Compilation before runtime

Compiler vs. Runtime Errors: Method Calls



* Start at the Actual Type at run time. If it doesn't exist in Actual Type, trace up the inheritance tree to find the nearest inherited method definition

Compiler vs. Runtime Errors: Casting





Practice : Pair



sli.do

#cse123

What is the result of the following code?

```
Teacher socrates = new Teacher("Socrates", 2400);  
System.out.println(socrates.getEmployeeName());  
System.out.println(socrates.getYearsExperience());
```

- A. Compiler Error
- B. Runtime Error
- C. No error– runs to completion



Practice : Pair

[sli.do](#)

#cse123

What is the result of the following code?

```
Employee anthony = new Chef("Anthony Bourdain");  
System.out.println(anthony.getEmployeeName());  
anthony.cookFood("shrimp");
```

- A. Compiler Error
- B. Runtime Error
- C. No error— runs to completion



Practice : Pair

sli.do

#cse123

What is the result of the following code?

```
Employee anthony = new Chef("Anthony Bourdain");  
System.out.println(anthony.getEmployeeName());  
Chef c = (Chef) anthony;  
c.cookFood("shrimp");
```

- A. Compiler Error
- B. Runtime Error
- C. No error— runs to completion



Practice : Pair

sli.do

#cse123

What is the result of the following code?

```
Employee anthony = new Chef("Anthony Bourdain");  
System.out.println(anthony.getEmployeeName());  
Teacher t = (Teacher) anthony;  
t.getYearsExperience();
```

- A. Compiler Error
- B. Runtime Error
- C. No error— runs to completion

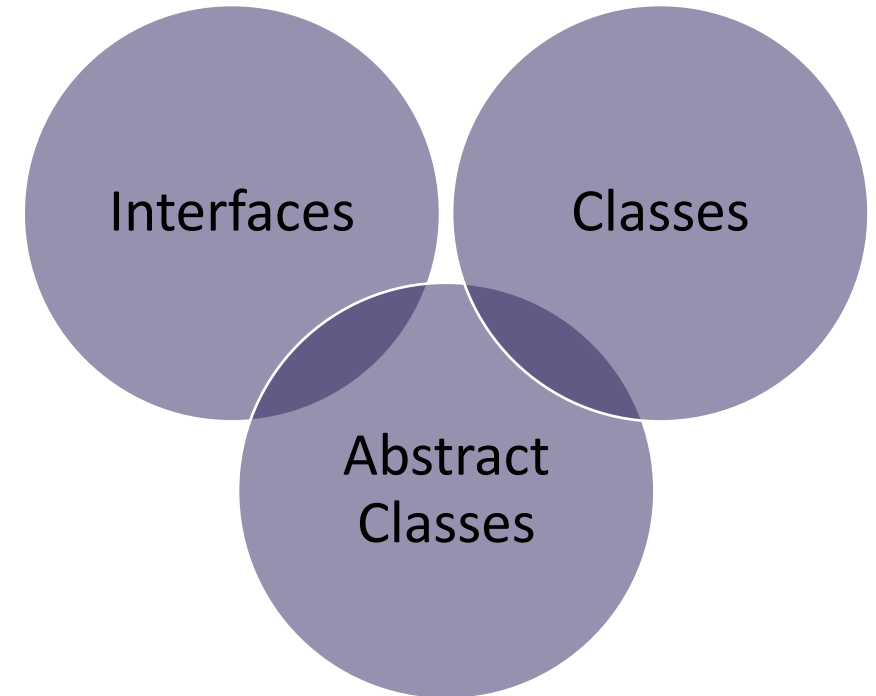
Lecture Outline: Abstract Classes

- Polymorphism
- Compiler vs. Runtime Errors
- **Abstract Classes** ◀
- Academic Honesty Policy

Abstract Classes

- Mixture of Interfaces and Classes

- Interface similarities:
 - Can contain (abstract) method declarations
 - Can't be instantiated
- Class similarities:
 - Can contain method implementations
 - Can have fields
 - Can have constructors



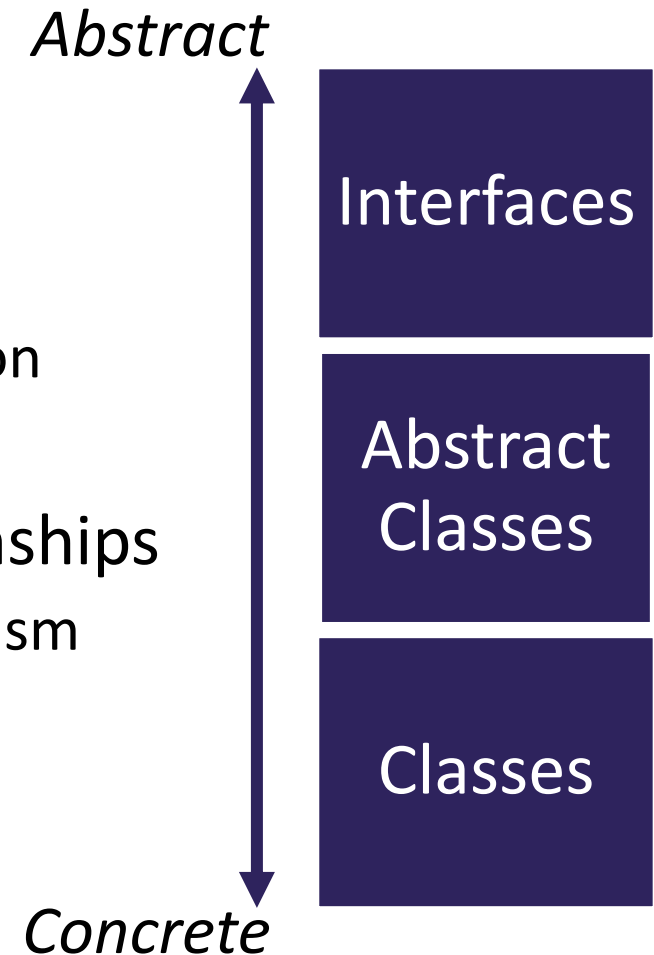
- Is there identical / nearly similar behavior between classes that shouldn't inherit from one another?

Shape Example: Tasks

- Add an abstract `getName` method to the `Shape`.
 - Add implementations of `getName` to `Square` and `Circle` that return "Square" and "Circle".
- Add a method `isEmpty` to `Shape` that tells you whether the shape is empty (has zero area) or not.
 - Hint: you will need to call `getArea`, but we've included a slight trap related to `getArea` in the starter code!
- Implementing `isEmpty` by calling `getArea` works fine as is, but suppose we wanted to implement it in `Circle` directly. How could we do this just by looking at the fields of the `Circle`? Implement it this way by overriding `isEmpty` in `Circle`.
- Override `toString` in `Shape` to return a similar message to what the `Client` prints in the starter code:
 - Hint: your `toString` can call abstract methods!
- Rewrite the `Client` class to use the new `toString` on shapes.

Advanced OOP Summary

- Allow us to define differing levels of abstraction
 - Interfaces = high-level specification
 - What behavior should this type of class have
 - Abstract classes = shared behavior + high-level specification
 - Classes = individual behavior implementation
- Inheritance allows us to share code via “is-a” relationships
 - Reduce redundancy / repeated code & enable polymorphism
 - Still might not be the “best” decision!
 - Interfaces extend other interfaces
 - (abstract) classes extend other (abstract) classes



- You're now capable of designing some pretty complex systems!

Design in the “real world”

- In this course, we’ll always give you expected behavior of the classes you write
 - Often not the case when programming for real
 - Clients don’t really know what they want (but programmers don’t either)
- My advice:
 - Clarify assumptions before making them (do I really want this functionality?)
 - **There’s no one right answer**
 - Weigh the options, make a decision, and provide explanation
 - Iterative development: make mistakes and learn from them
 - Be receptive to feedback and be willing to change your mind

Interface vs. Implementation

- Interface: what something *should* do
- Implementation: *how* something is done
- These are different!
- Big theme of CSE 123:
choose between different implementations of same interface