LEC 01

CSE 123
Inheritance; Polymorphism

Questions during Class?
Raise hand or send here

sli.do #cse123

BEFORE WE START

Talk to your neighbors:

Plans for the weekend?

Instructors: Ziao Yin

TAs: Nichole Packard Trien Eeshani Chris

Coming up…

- ❓ Complete the Introductory Survey
  - This helps us gather data about the students taking our classes and their backgrounds, to inform future offerings.
- 🙋‍♀️ Review Section0.5 in Ed
  - Includes material covered in cse121 and 122 to help review and jog your memory!
- 💻 The IPL opens Monday, Jun 30
  - Schedule posted soon
- 🔎 Creative Project 0: Search Engine out now
  - Due Wednesday, Jul 2, 11:59pm

Grading

Grades should reflect your proficiency in the course objectives

- All assignments will be graded **E (Excellent), S (Satisfactory), or N (Not yet)**
  - Under certain circumstances, a grade of U (Unassessable) may be assigned
- Final grades will be assigned based on the **amount of work at each level**

- See the syllabus for more details

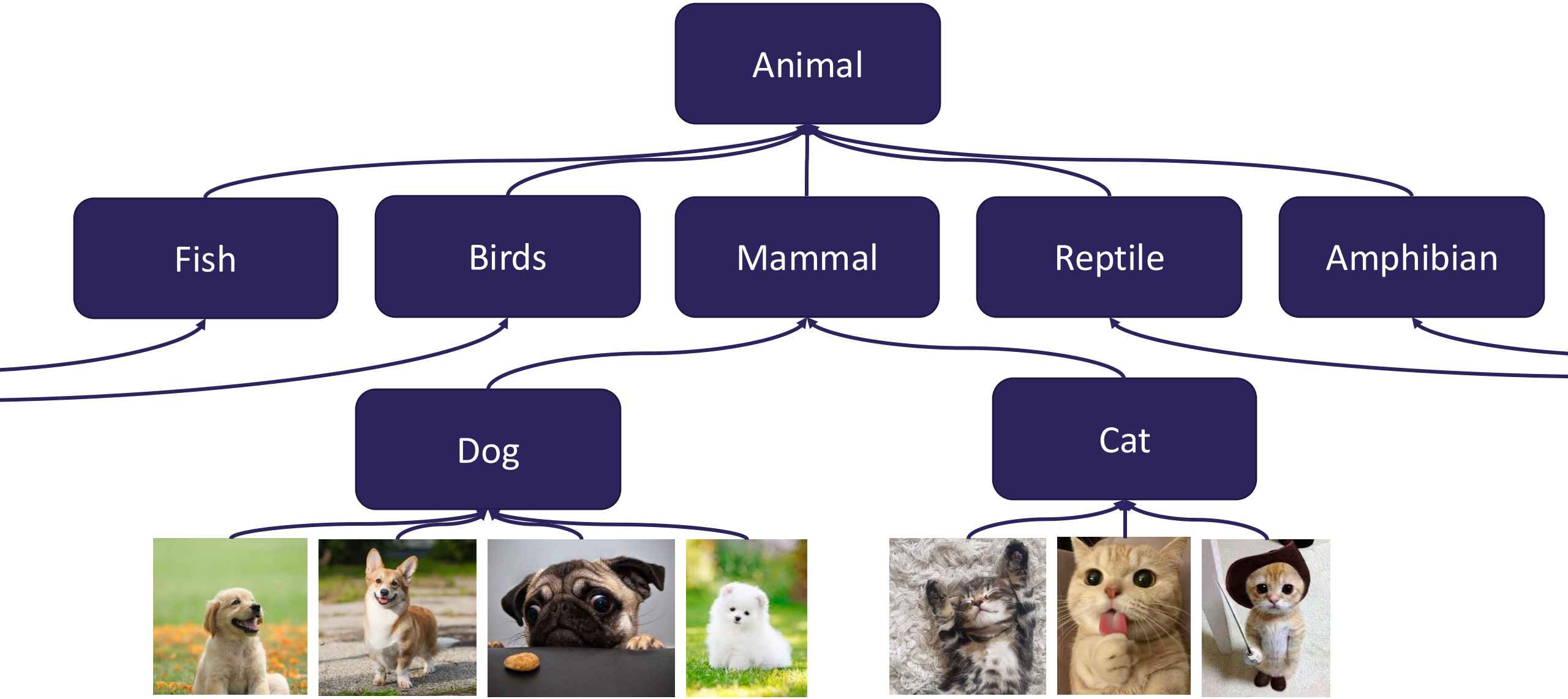
Lecture Outline

- **Inheritance**
- Polymorphism
  - Declared vs. Actual Type
  - Compiler vs. Runtime Errors

Inheritance

- Connect together a “subclass” and “superclass”
  - Borrow / “inherit” code to reduce redundancy
  - `super()` keyword can be used just like `this()`
- Syntax: `public class Subclass extends Superclass`
- Should Represent “is-a” relationships
  - `public class Chef extends Employee`
  - `public class Server extends Employee`
- In Java, all objects implicitly inherit from the Object class
  - toString(), equals(Object), etc.

Is-a Relationships



Lecture Outline

- Inheritance
- **Polymorphism**
  - Declared vs. Actual Type
  - Compiler vs. Runtime Error0s

Polymorphism

- `DeclaredType x = new ActualType()`
  - All methods in `DeclaredType` can be called on x
  - We've seen this with interfaces (`List<String>` vs. `ArrayList<String>`)
  - Can also be to inheritance relationships

```
Animal[] arr = {new Dog(), new Cat(), new Bear()};
for (Animal a : arr) {
    a.feed();
}
```

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) a).bark();            // No more CE
```

  - Runtime error: attempting to cast to an invalid `DeclaredType*`

```
Animal a = new Fish();
((Dog) a).bark();            // Can't cast -> RE
```

  - Order matters! Compilation before runtime

Declared Type and Actual Type

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

```
Animal bucky = new Dog("Bucky");
```

Declared Type: Animal
Actual Type: Dog

Can call methods that makes sense for EVERY Animal
If Dog overrides a method, uses the Dog version

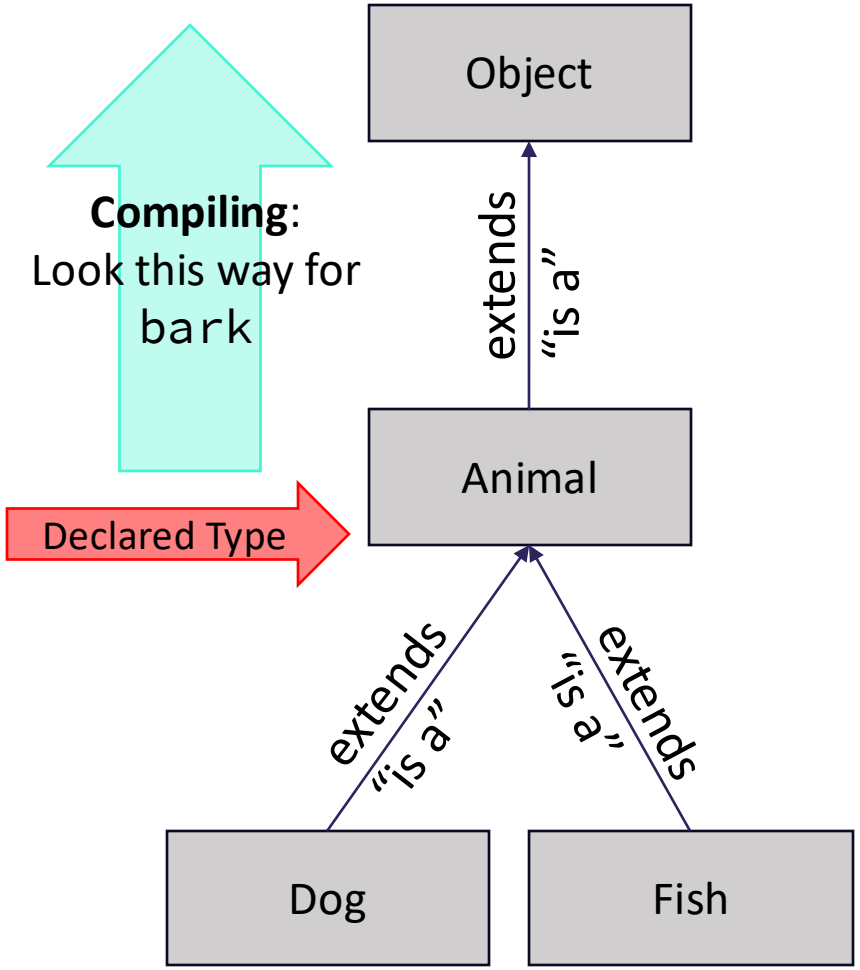
```
Dog bucky = new Dog("Bucky");
```

Declared Type: Dog
Actual Type: Dog

Can call methods that makes sense for EVERY Dog
If Dog overrides a method, uses the Dog version

Inheritance and Method Calls

```
Animal bucky = new Dog();
bucky.bark();
```



When compiling:

Can we *guarantee* that the method exists for the declared type?

Does the declared type or one of its super classes contain a method of that name?

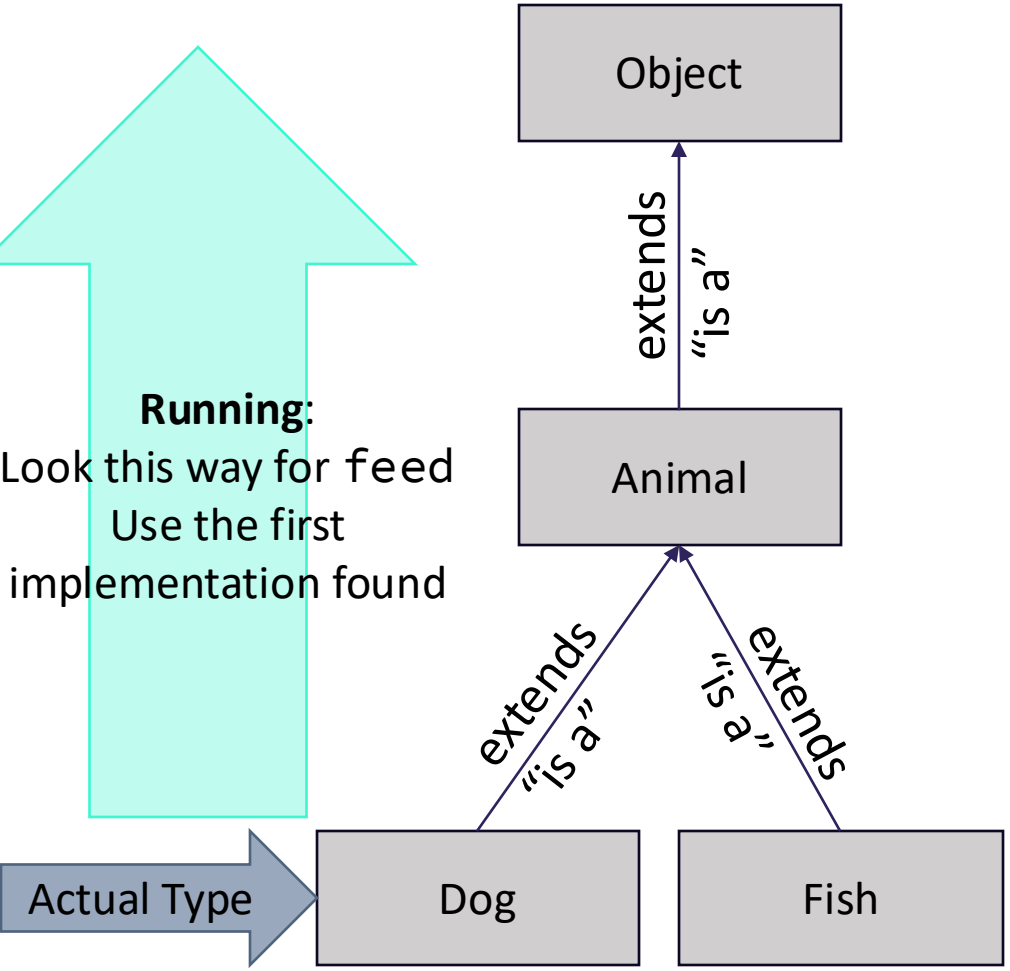
If not… Compile Error!

In this example:

When compiling, neither Animal nor Object have a bark method, so we have a compile error!

Overrides and Method Calls

```
Animal bucky = new Dog();
bucky.feed();
```



When running:

Use the *most specific* version of the method call starting from the actual type.

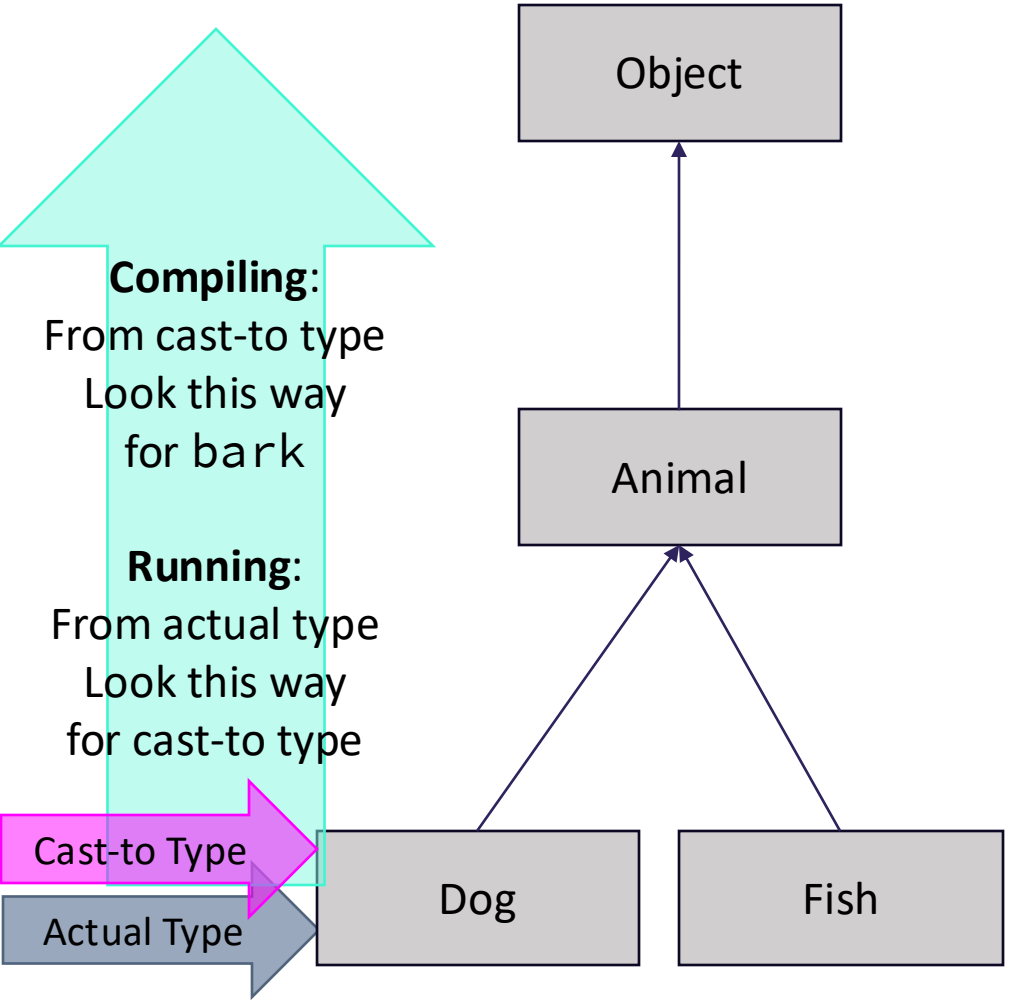
Start from the actual type, then go "up" to super classes until you find the method. Run that first-discovered version.

In this example:

If the Dog class overrides feed, then we'll use the implementation in Dog. Otherwise we'll use the one in Animal

Casting and Method Calls

```
Animal bucky = new Dog();
((Dog) bucky).bark();
```



When compiling:

Can we *guarantee* that the method exists for the Cast-to type?

Does the Cast-to type or one of its super classes contain a method of that name?

If not… Compile Error!

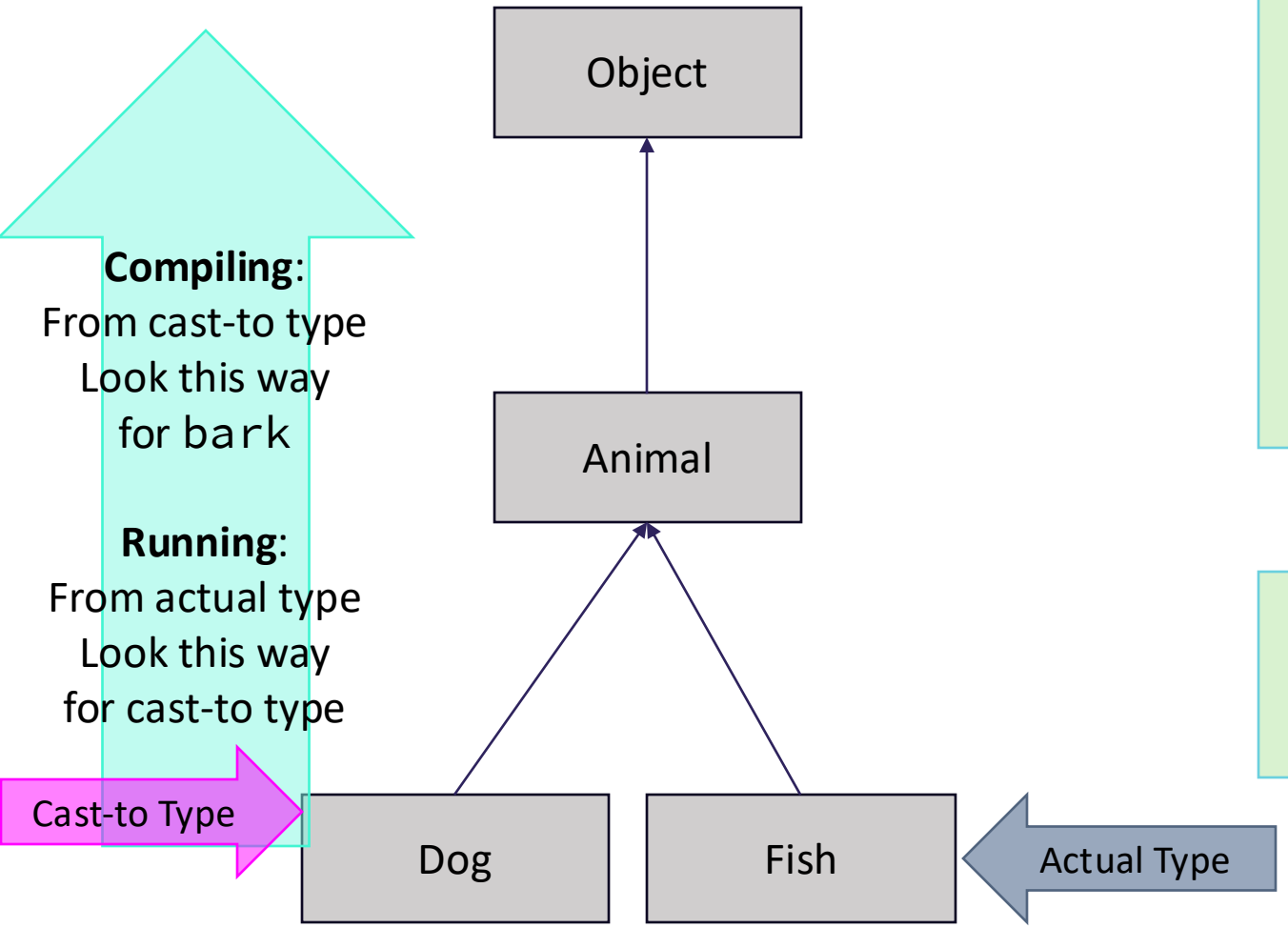
When Running:

Check that the Cast-to Type is either the Actual Type, or one of its super classes

This example has no error

Casting and Method Calls

```
Animal bucky = new Fish();
((Dog) bucky).bark();
```



When compiling:

Can we *guarantee* that the method exists for the Cast-to type?

Does the Cast-to type or one of its super classes contain a method of that name?

If not… Compile Error!

When Running:

Check that the Cast-to Type is either the Actual Type, or one of its super classes

This example has a runtime error

Compiler vs. Runtime Errors

With the following declaration and initialization:

```
DeclaredType name = new ActualType();
```

If we call:

```
name.method();
```

Does **DeclaredType** contain **method**?

- Yes → Execute **method** of **ActualType**
- No → Compiler Error

```
((CastToType) name).method();
```

Is it conceivable that an instance of **DeclaredType** could be a **CastToType**? *

*This is true if either **DeclaredType** or at least one of its subtypes can substitute for a **CastToType**

- No → Compiler Error
- Yes → Does **CastToType** contain **method**?
  - No → Compiler Error
  - Yes → Does **ActualType** cast to **CastToType**?
    - No → Runtime Error
    - Yes → Execute **method** of **ActualType**