Inheritance

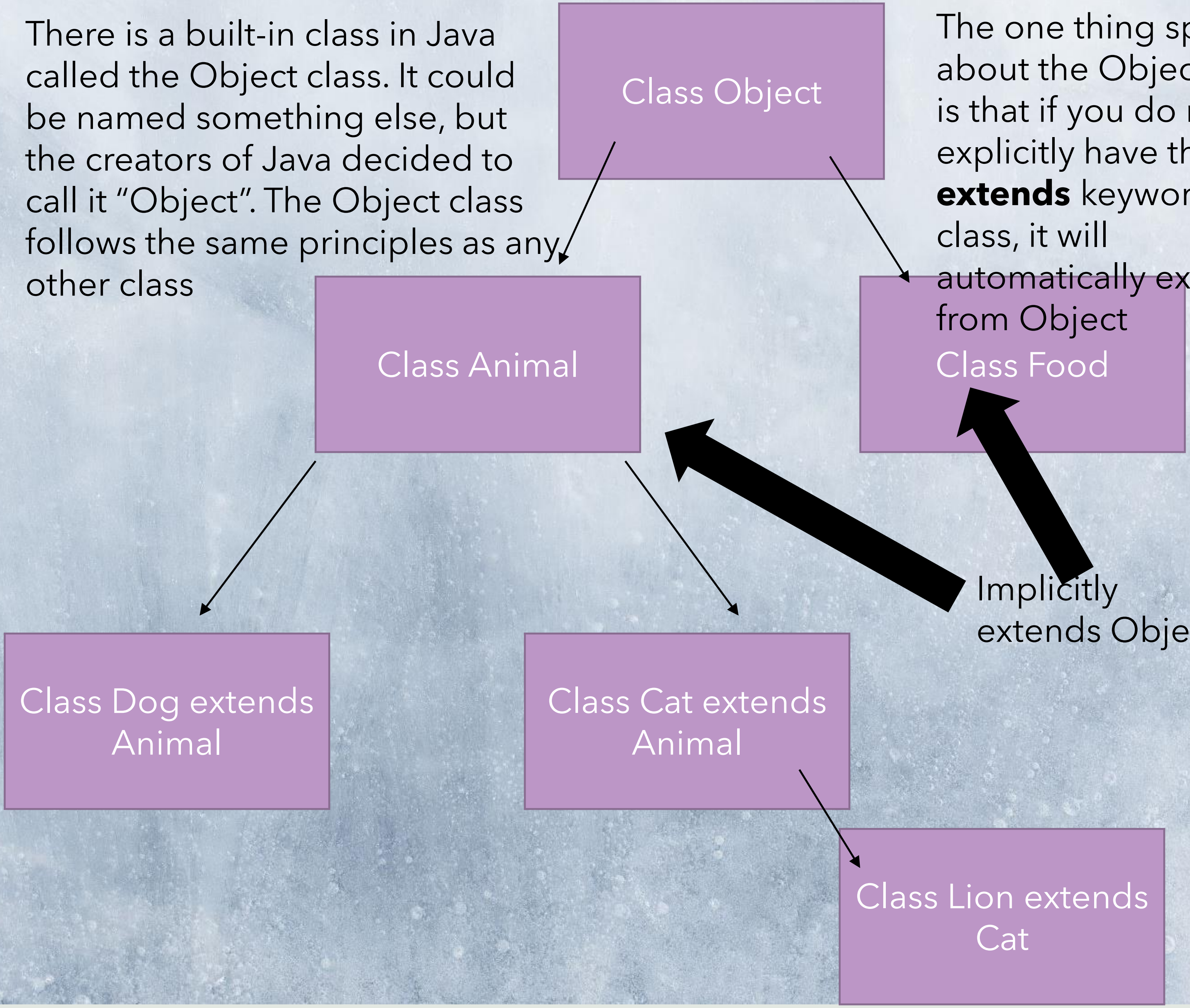
- The point of inheritance is to have classes inherit the properties and behaviors of another class
- This allows us to prevent code duplication when multiple classes use the same properties and methods
- For example, there are properties and behaviors universal to all types of Animal
- Instead of defining the exact same properties and behaviors over and over again for each specific type of Animal, we can just inherit from a base class called Animal

Terminologies

- Superclass/base-class/parent class = A class higher in the hierarchy (Animal)
- Subclass/Child class = A class lower in the hierarchy (Dog and Cat)
- Phrases:
- “Lion is a subclass of cat”
- “Dog and Cat are subclasses of Animal”
- “Animal is a subclass of Object”

There is a built-in class in Java called the Object class. It could be named something else, but the creators of Java decided to call it "Object". The Object class follows the same principles as any other class

The one thing special about the Object class is that if you do not explicitly have the **extends** keyword on a class, it will automatically extend from Object



Subclasses inherit the properties and behaviors of not only the parent, but also the grandparent/great-grandparent …

Upcasting/downcasting (revisited)

- Whenever we create classes, we also create **types** associated with those classes
- For example, Animal, Dog, Cat, and Lion are now variable types (in addition to being types of objects)
- Upcasting and downcasting are all about converting from one type of variable to another
- Upcasting (implicit): going from a more specific type to a less specific type (going from Dog to Animal, we are going up the hierarchy, which is also being less specific)
- Downcasting (explicit): going from a less specific type to a more specific type (Lion is more specific than Cat, we are going down the hierarchy)

Polymorphism

- Poly means many
- Morph means form
- Polymorphism = many forms
- What we are specifically talking about is method overriding or method overloading
- Method overriding = **runtime polymorphism** (the JVM checks to see what type of object the variable is actually pointing to as the program is running)
- Method overloading = **compile-time polymorphism** (the compiler takes our source code and converts it into bytecode instructions that contain execution paths for the JVM when it is running the bytecode. So, when we do method overloading, it knows what execution path to take (aka the particular overloaded method) whenever the program is compiled)

Object Class

- There is a built-in class in Java called the Object class. It could be named something else, but the creators of Java decided to call it “Object”. The Object class follows the same principles as any other class
- The one thing special about the Object class is that if you do not explicitly have the **extends** keyword on a class, it will automatically extend from Object
- Whatever methods and/or properties defined in the Object class by the creators of the Object class will be inherited by all other classes

Object class Methods

- toString() method
- hashCode() method
- equals() method:
- When we compare values of strings, we use the equals() method. But, the equals method does not actually come from the String class. The String class is actually overriding the equals method from the Object class (to compare characters in two different String objects to see if all of them are the same)