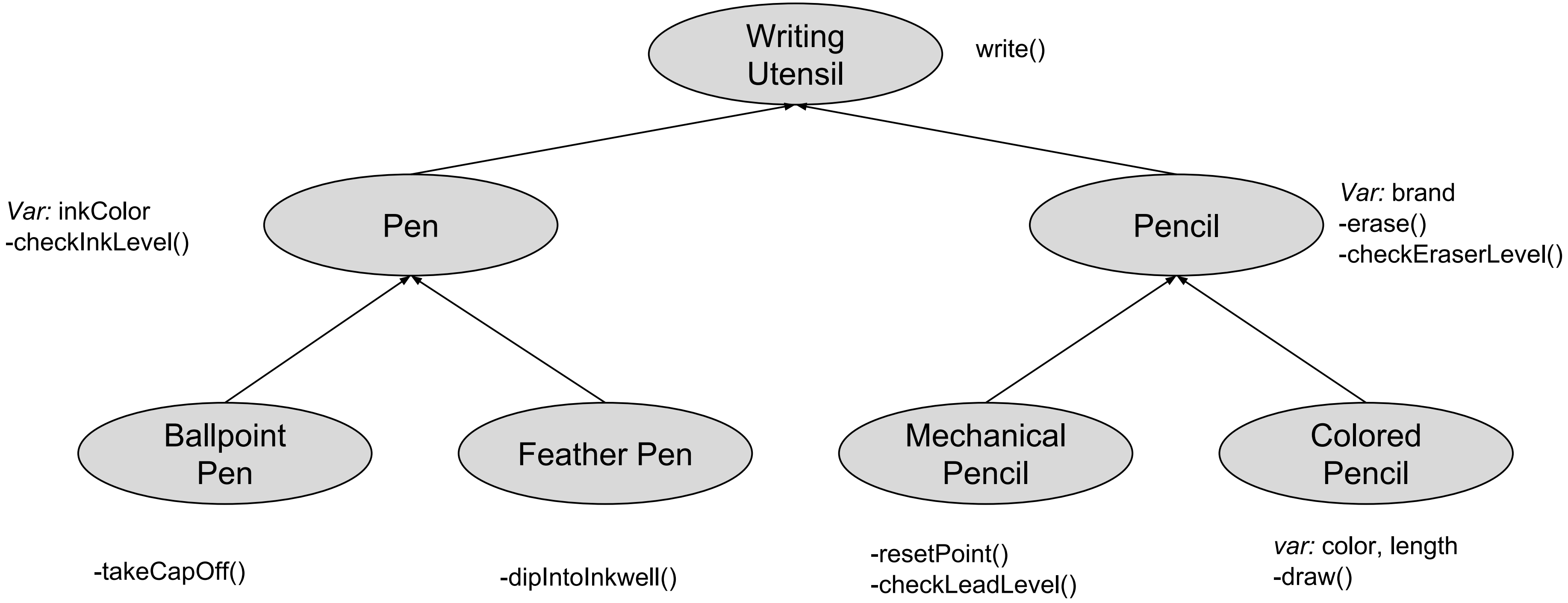
Interfaces

It’s all about the memes

Quick Review of Objects and Classes

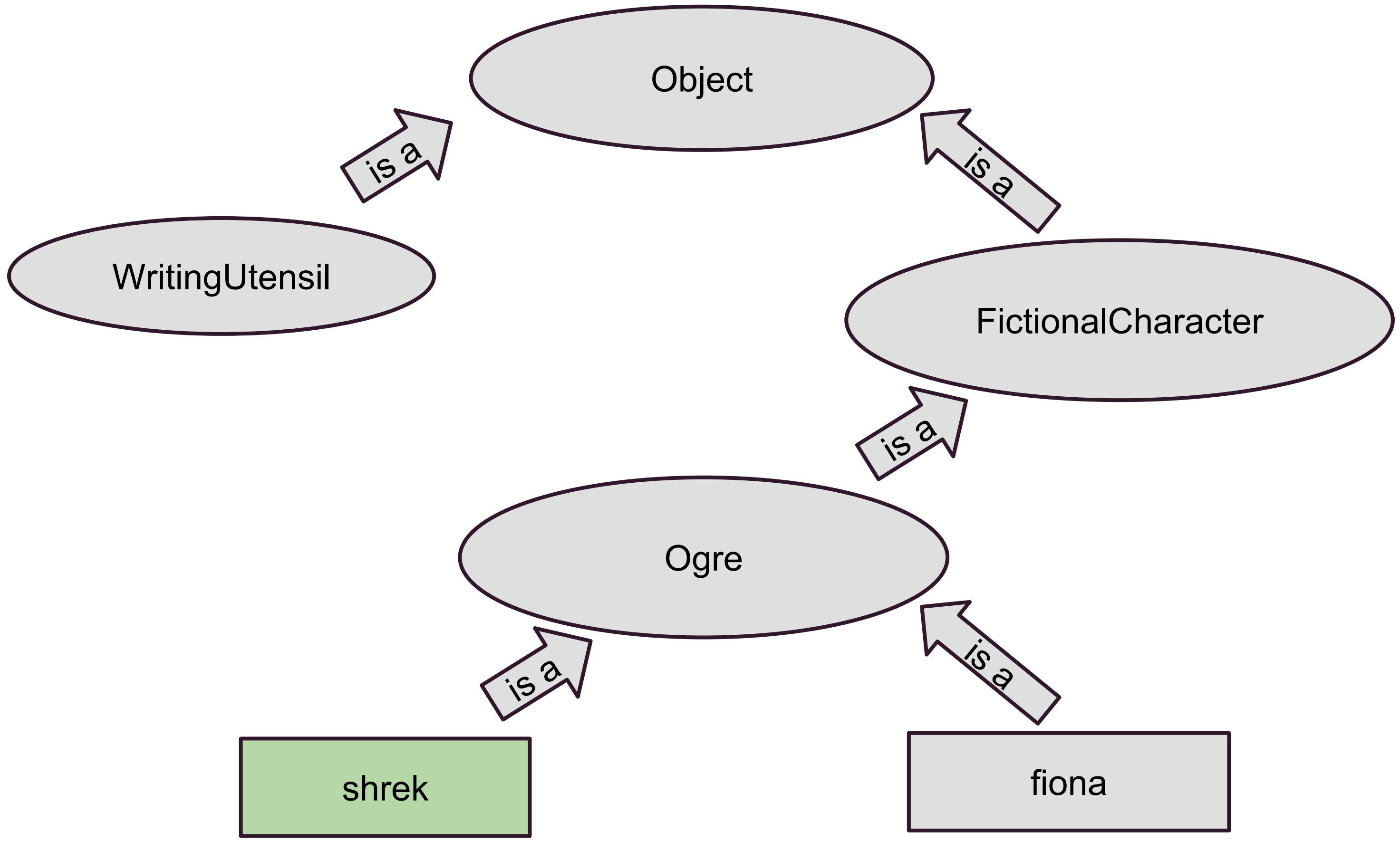
- Java is object-oriented
- All data is stored in objects, which in turn are manifestations of classes

Conceptual Example: Writing Utensil



Polymorphism

- Simply the relationship that describes an object in terms of a hierarchical structure
- Use the “IS-A” test


Object
WritingUtensil
is a
is a
FictionalCharacter
is a
Ogre
is a
is a
shrek
fiona

Object Example

```
Ogre shrek = new Ogre(“green”);
```

The object `shrek` is an instance of the class `Ogre`, which in turn is a subclass of `FictionalCharacter`, which in turn is a subclass of `Object`.

Type Casting

What if I told you this is *valid*:

```
Ogre shrek = new Ogre(“green”);
FictionalCharacter swamp = (FictionalCharacter) shrek;
```

But this is *invalid*:

```
FictionalCharacter farquaad = new FictionalCharacter();
Ogre fiona = (Ogre) farquaad;
```

Multiple Inheritance

- Many languages (C++, Python, JavaScript [sort of]) allow you to extend multiple classes - this is called **multiple inheritance**
- Scumbag Java, on the other hand, can only inherit from *one* superclass, but similar functionality can be achieved by using multiple class-like structures called **interfaces**

Interface Example

```
public interface InterfaceName{

    //any number of final, static variables
    //any number of abstract method declarations

}
```

Interface Example

```
public Interface AlliumVegetable{

    public int getNumLayers();
    public void setNumLayers(int num);
    public boolean isDelicious();

}
```

Fun Fact:
Allium style vegetables are veggies like onions, shallots, leeks, and shives.

Interfaces Implementation Example

```
public class Onion implements AlliumVegetable{

    //Onion variables
    //Onion methods
}
```

Use the keyword *implements* when using an interface

What's wrong?

Correct Implementation

All interface methods/variables must be implemented

```
public class Onion implements AlliumVegetable{

      //Onion variables
      //Onion methods

      public int getNumLayers(){
            return numLayers;
      }
      public void setNumLayers(int num){
            this.numLayers = num;
      }
      public boolean isDelicious(){
            return false;
      }

}
```

This weird trick...

- Let’s assume that both Onion and Shrek implement interface AlliumVegetable

This works!

```
AlliumVegetable strangeFruit = new Onion();
AlliumVegetable shrek = new Shrek();
```

But this **doesn’t...**

```
AlliumVegetable veggie = new AlliumVegetable();
```

- Even though AlliumVegetable is an interface, it can be used as a type only when instantiated using a class that implements it, but not on its own.

A few things to note about interfaces

Both of these work:

```
strangeFruit.getNumLayers();
shrek.getNumLayers();
```

However, the way that each method acts is completely different because each class implements it in a different way.

Classes & Multiple Interfaces

```
public class BillGates extends Human implements CEO,
Philanthropist{}
```

- Here we have a class that is a subclass of Human and implements two interfaces.
- Normally one would not be able to inherit from both `CEO` and `Philanthropist` but they are interfaces
- `BillGates` can choose to define its own methods and override those of its superclass.
- It must implement whatever the interfaces demand.

Scumbag Java

Want a list in Java? You might try:

```
    List fiona = new List();
```

But `List` is not a proper class. It’s not even an abstract class. It’s an interface. Interfaces have no constructors and you cannot make instances of them.

The ArrayList

Use `ArrayList<T>` instead:

```
public class ArrayList<T> implements List;
```