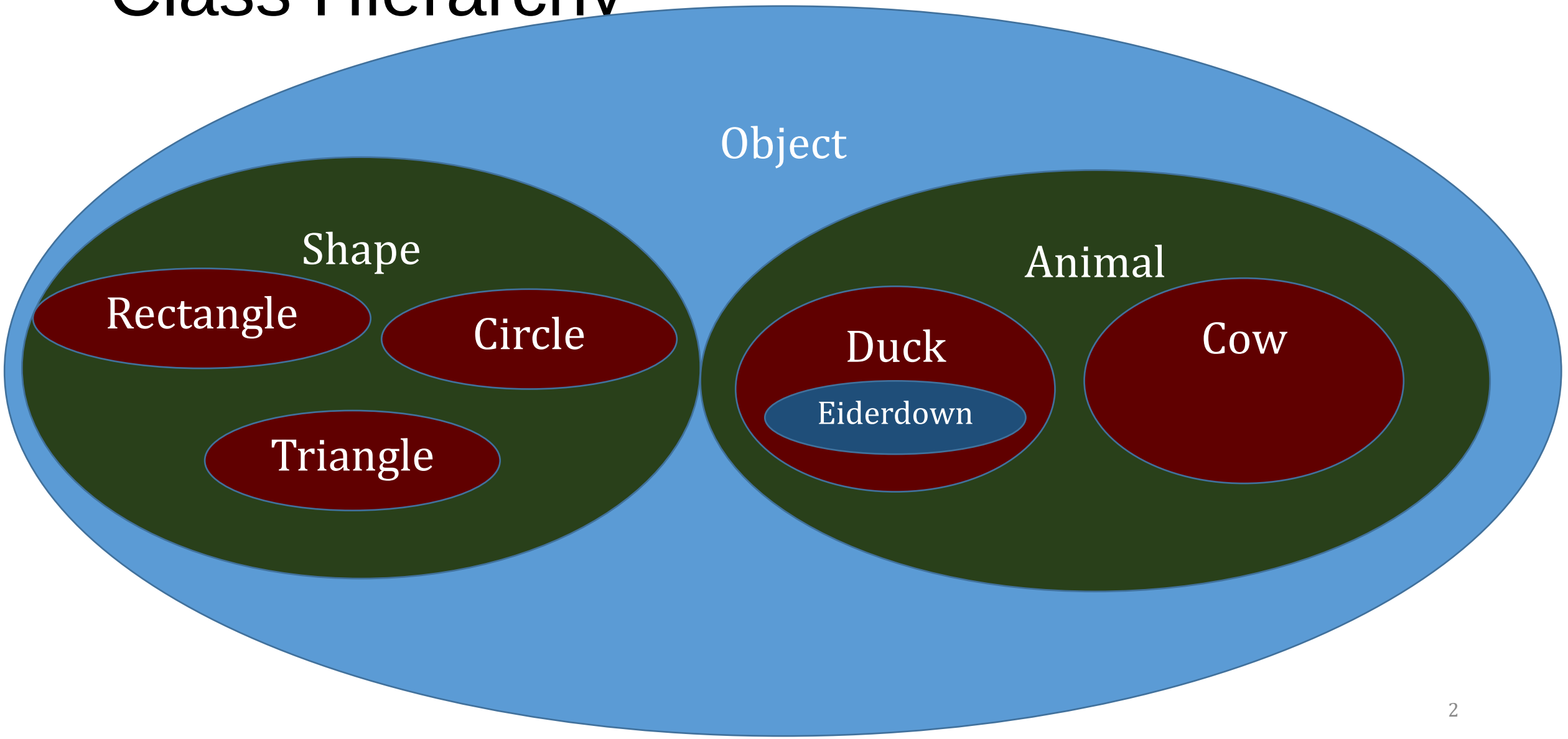

The Universal Class

Class Hierarchy



The Universal "Object" Class

- If you define a class with no "extends" keyword, Java implicitly adds "extends Object"
- The Object class is defined in the Java library
- Therefore, all objects are descendants of the Object class
  - All objects inherit the Object class fields and methods!
  - All classes can override Object class methods

Object.toString

- The library implementation is:

```
public String toString() {
        return getClass().getName() + '@' +
                        Integer.toHexString(hashCode());
}
```

- Overriding toString is recommended

Object.getClass()

- Returns an object of type "Class<T>", where T is the class or a sub-class of the reference object

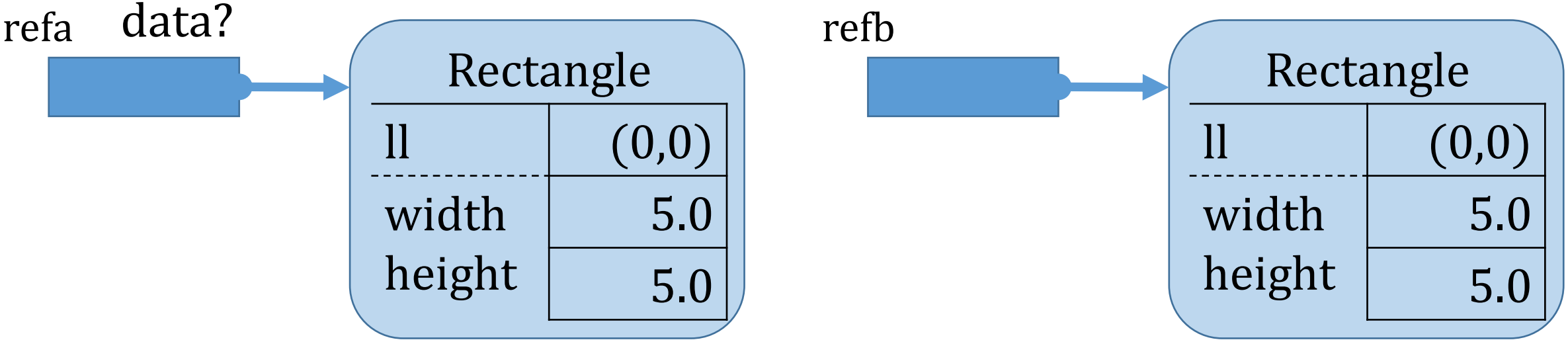
```
Number n = 0;
Class<? extends Number> c = n.getClass();
```

Generic type specification not required

- The returned Class object is the **dynamic type** of the reference object.
- Look up Class in the java library… use this to get:
  - class name, package name, methods, fields, parent class, etc.

Object.equals(Object obj)

- In Java, `refa == refb` returns true if refa and refb are referencing the exact same object
- Suppose refa and refb point to two different objects with similar data?



- `refa==refb` is "false", `refa.equals(refb)` can return "true"!

equals properties

- Reflexive: a.equals(a) should always return true
- Symetric: if a.equals(b) is true, then b.equals(a) should be true
- Transitive: if a.equals(b) and b.equals(c) are true, then a.equals(c) should be true
- Consistent: if a.equals(b) is true, and a and b do not change, then a.equals(b) should return true
- Not null: For any non-null reference, a, a.equals(null) should return false

Object.equals implementation

```
public boolean equals(Object obj) {
        if (obj==null) return false;
        return (obj == this); //pessimistic implementation!
}
```

Less Pessimistic Override

```
public class Rectangle extends Shape {
        …
@Override public boolean equals(Object obj) {
        if (!(obj instanceof Rectangle)) return false;
        Rectangle r = (Rectangle) obj; // Explicit down-cast
        return super.equals(r) &&
                r.width==this.width &&
                r.height == this.height);
}
```

hashCode

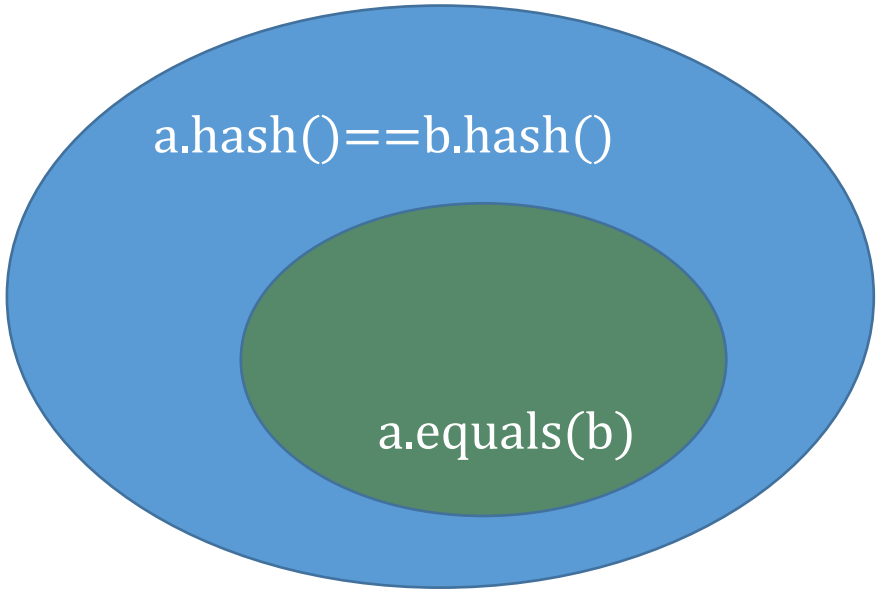
Special Topic 15.1

- More later, but for now, think of the hashCode function as something which returns a semi-unique integer for each object
- Hash codes are used for quick look-up of an object
- For now, we only need worry about the rule:

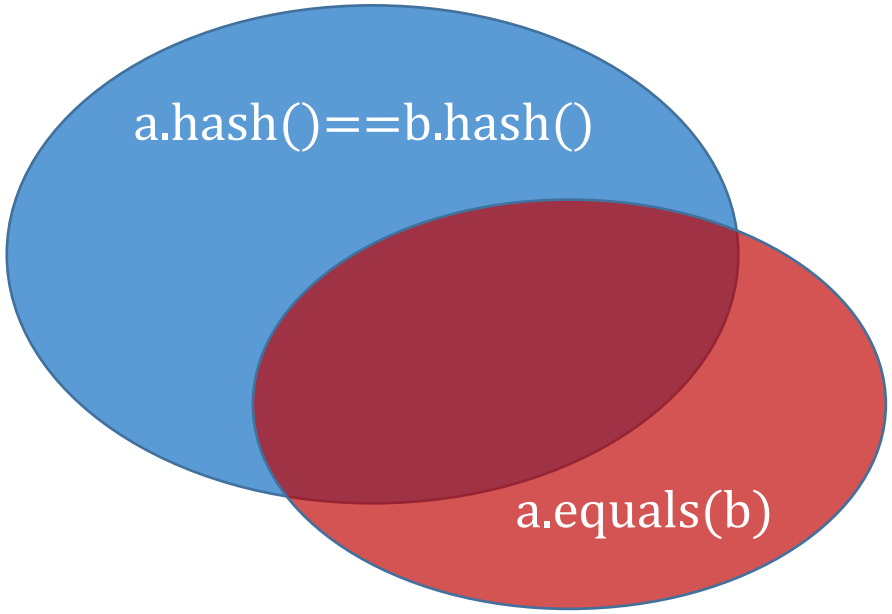
  "If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result."

hashCode vs. equals

This is OK… Java likes this…



This is NOT OK… java has a problem



Example hashCode

```
@Override public int hashCode() {
        Double widthBoxed = width;
        Double heightBoxed = height;
        return super.hashCode() +
                widthBoxed.hashCode() +
                heightBoxed.hashCode();
}
```