Well Developed Classes w/ OOP

Well Developed Classes

- By convention, classes should be well developed
- What does this mean?
- Ease of use

Well Developed Classes

- Encapsulation
- toString()
- Comparable (+ ators)
- Copyable
- Robust Exceptions

Aside: toString()

- When we write a toString()
- We are *overriding* an existing method
- This is different from overloading
- This comes from the parent class (object) class in this case

Complex Comparable

- We’ve seen simple comparable
- -1, 0, +1
- This can be strung together for more complex ordering!

- I.e. sort by ddNumber, then by title
- Let’s do it!

Complex Comparable

- Where should we put this?
- First compare ddNumbers
- How do we compare the Strings?
- Let’s check the documentation

Complex Comparable

- Comparable is for *natural ordering*
- What if we wanted to define another way of sorting our class?

- Comparator Interface
- Used for arbitrary ordering
- Defined *elsewhere*

Copyable

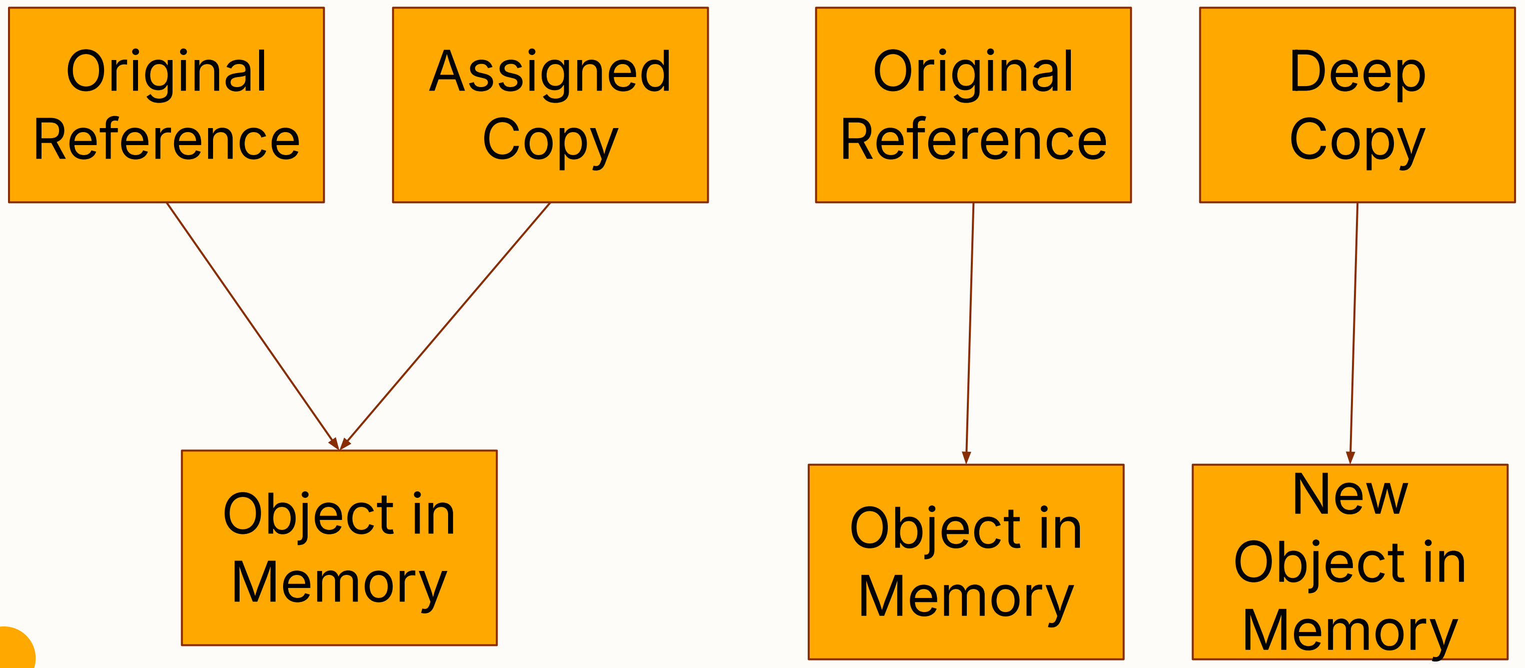
- Good classes should be easy to use and copy

- Special constructors to make this easier
- Copy Constructors

Why Copy Constructors

- What happens if we don’t use a copy constructor
- Book b1 = new NonFictionBook(...);
- Book b2 = b1;
- b2.setDDNumber(90.12)

Shallow vs Deep Copying



Let’s revisit some old ideas

- Type casting
- What happens if we do a bad cast?
- What makes something a bad cast?

Casting

```
Book b1 = new FictionBook(
    14.01,
    "Twilight"
);
```

Can I cast b1 to a NonFictionBook?

Uh oh!

- ClassCastException

- Occurs when we try to cast to a subclass that our object is not an instance of
- Let’s look at the documentation

Instantiation

- How can I avoid these exceptions?
- In other words, how can I verify the instance of an object?

- instanceof keyword!

instanceof

- Used to verify instantiation of an object
- All lowercase, all one word
- Often used in conditionals

- if (b1 instanceof FictionBook)