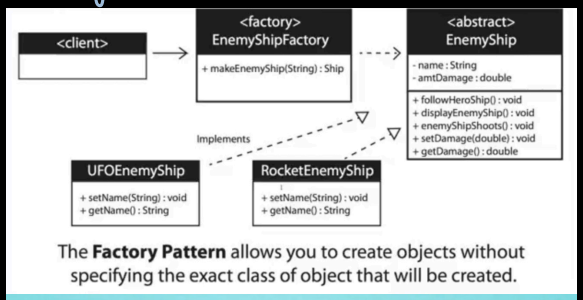
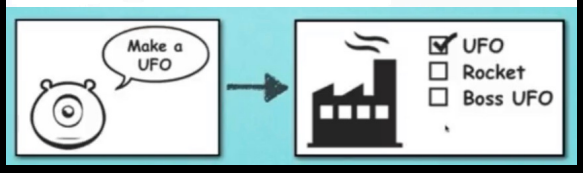
Factory is a method





What is the Factory Pattern?

- When a method returns one of several possible classes that share a common super class
  - Create a new enemy in a game
  - Random number generator picks a number assigned to a specific enemy
  - The factory returns the enemy associated with that number
- The class is chosen at run time

When to Use a Factory Pattern?

- When you don't know ahead of time what class object you need
- When all of the potential classes are in the same subclass hierarchy
- To centralize class selection code
- When you don't want the user to have to know every subclass
- To encapsulate object creation

enemyFactory

enemyObj

if input == X then produce objA

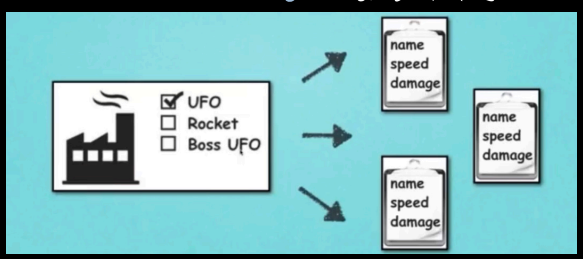
if input == Y then produce objB

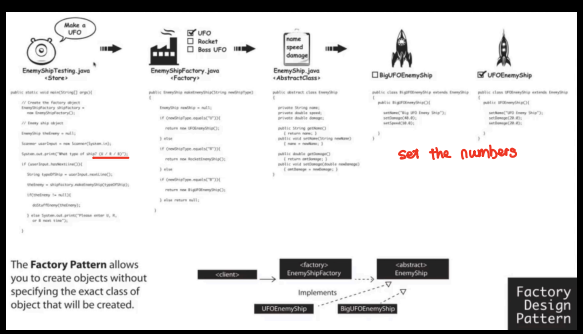
. . .

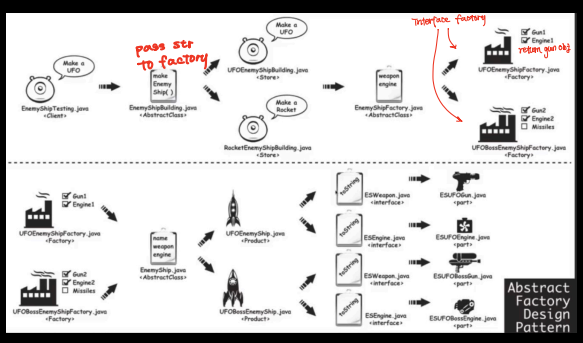
Abstract Factory

non-concrete abstract class

is an object







What is the Abstract Factory Pattern?

- It is like a factory, but everything is encapsulated
  - The method that orders the object
  - The factories that build the object
  - The final objects
  - The final objects contain objects that use the Strategy Pattern
    - Composition: Object class fields are objects

What can you do with an Abstract Factory?

- Allows you to create families of related objects without specifying a concrete class
- Use when you have many objects that can be added, or changed dynamically during runtime
- You can model anything you can imagine and have those objects interact through common interfaces
- The Bad: Things can get complicated

To show you the difference, here is a factory method in use:

```
class A {
    public void doSomething() {
        Foo f = makeFoo();
        f.whatever();
    }

    protected Foo makeFoo() {
        return new RegularFoo();
    }
}

class B extends A {
    protected Foo makeFoo() {
        //subclass is overriding the factory method
        //to return something different
        return new SpecialFoo();
    }
}
```

Factory method — one factory method

And here is an abstract factory in use:

```
class A {
    private Factory factory;

    public A(Factory factory) {
        this.factory = factory;
    }

    public void doSomething() {
        //The concrete class of "f" depends on the concrete class
        //of the factory passed into the constructor. If you provide a
        //different factory, you get a different Foo object.
        Foo f = factory.makeFoo();
        f.whatever();
    }
}

interface Factory {
    Foo makeFoo();
    Bar makeBar();
    Aycufcn makeAmbiguousYetCommonlyUsedFakeClassName();
}

//need to make concrete factories that implement the "Factory" interface here
```

Abstract Factory — has several factory methods

object