Class Composition Alternatives

Jim Medlock
January 2016

Goal

- Explore the differences between inheritance and composition in Java
    - Has-a vs. Is-a
    - Demonstrate advantages and disadvantages of the alternative implementations
- Utilize different Java capabilities to achieve similar results
    - Inheritance
    - Interfaces
    - Abstract Classes
- Adhere to Leskov Substitution Principle
    - "if S is a subtype of T, then objects of type T may be replaced with objects of type S”
    - Errors result when class X is derived from class Y and class X is used out of context in place of class Y
        - Subclassing Square from Rectangle and then using it in place of Rectangle creates an error when computing the dimensions since a Square has equal width and height

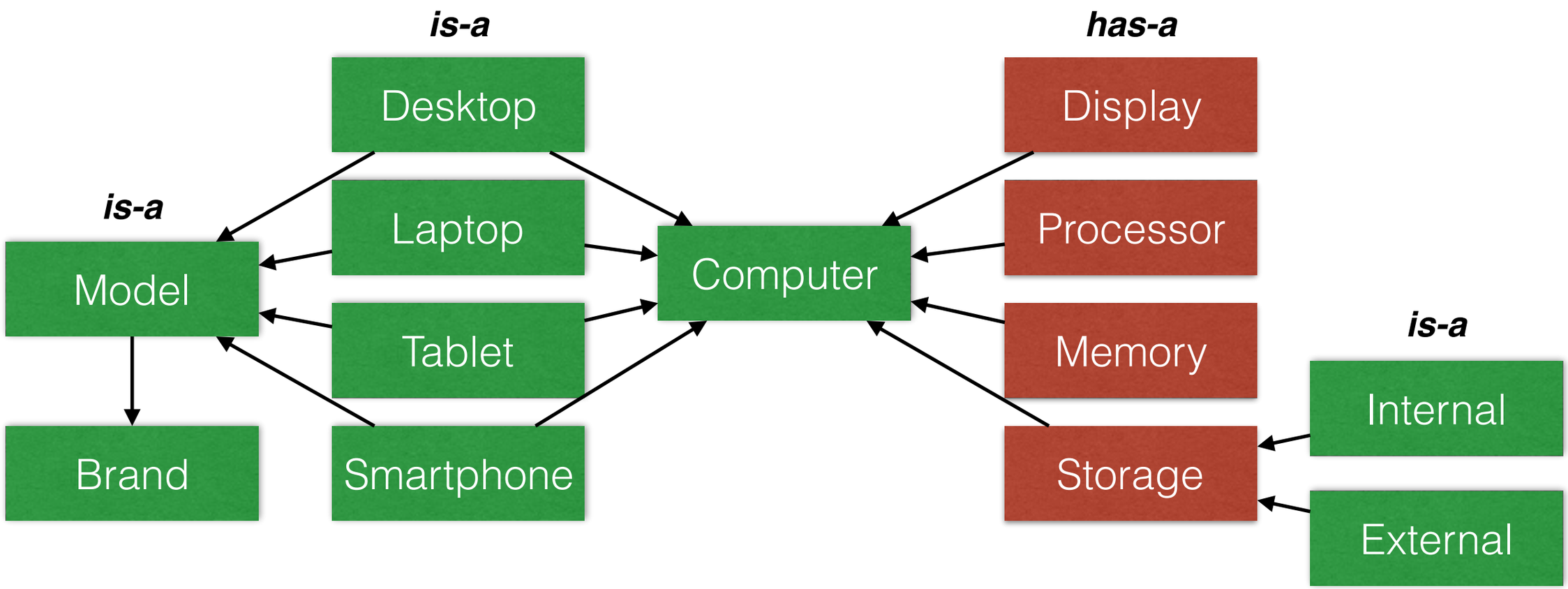
Inheritance vs. Composition

- Changes to inheritance-based object models can result in:
    - applications being difficult to change and exhibiting unintended side effects as the result of change.
    - increased change scope
- Composition is favored over inheritance
    - Since classes in Java don't support multiple inheritance Interfaces are more flexible because a class can implement multiple interfaces
        - No internal state required
    - Abstract classes provide better forward compatibility

Demonstration Application

- Model a generic computer including its features and functionality for a variety of “smart” devices including:
    - Desktop
    - Laptop
    - Smartphone
    - Tablet
- Develop three different implementations using inheritance, interfaces, and abstract classes to allow each to be compared and contrasted.
- Each solution uses its implementation to build the representation of a smartphone, tablet, desktop, and laptop. It then prints the attributes of each.

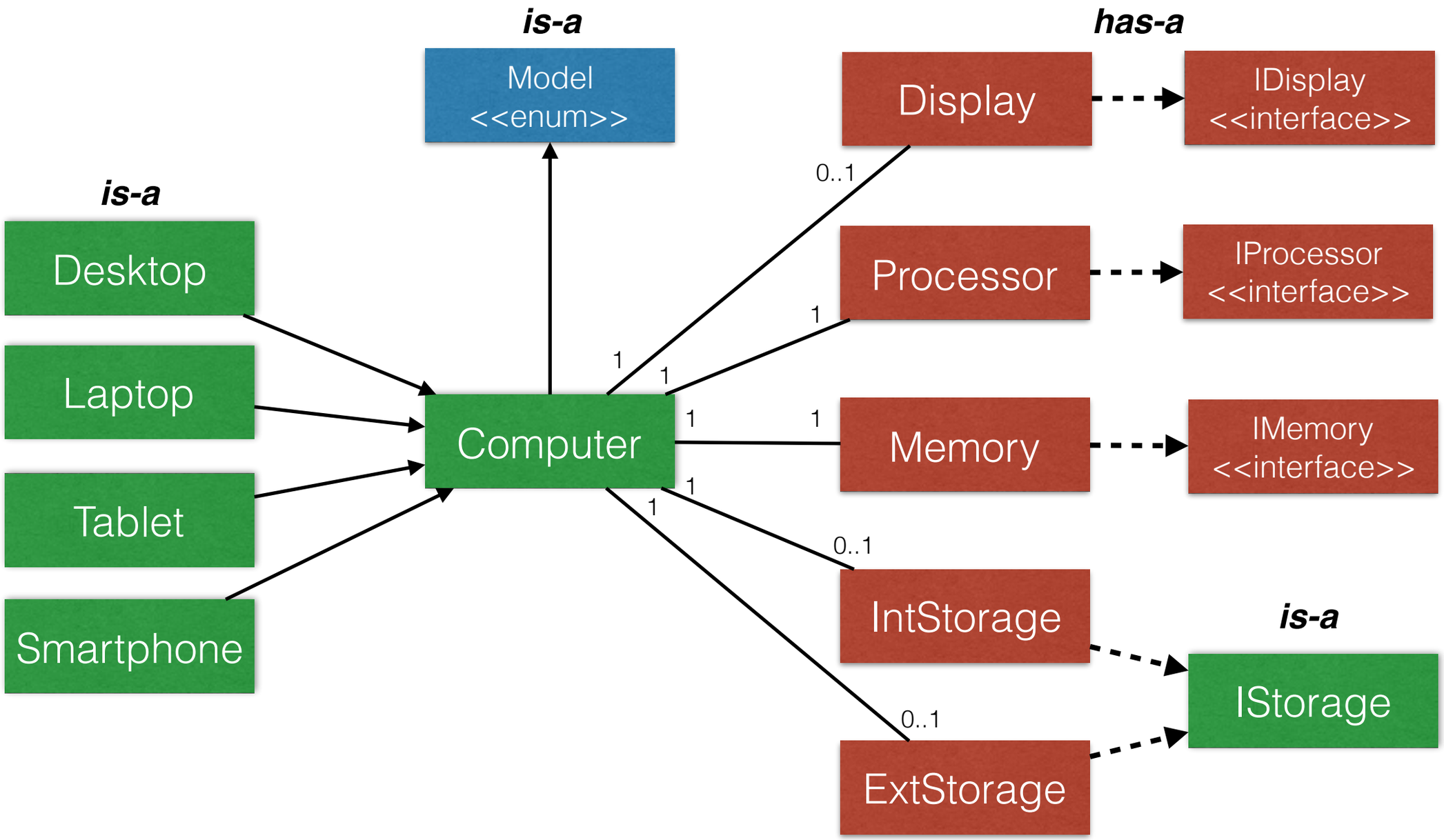
High-Level Object Relationships



An Example

- I own a smartphone
- It is an iPhone 6 Plus (model)
- It is manufactured by Apple (brand)
- It has a 5.5” display with 1080x1920 pixel resolution (display)
- It has a A8, dual-core processor running at 1.4 GHz (processor)
- It has 1GB of internal DDR3 memory (memory)
- It has 64GB of internal storage (storage)

Inheritance-Based Model

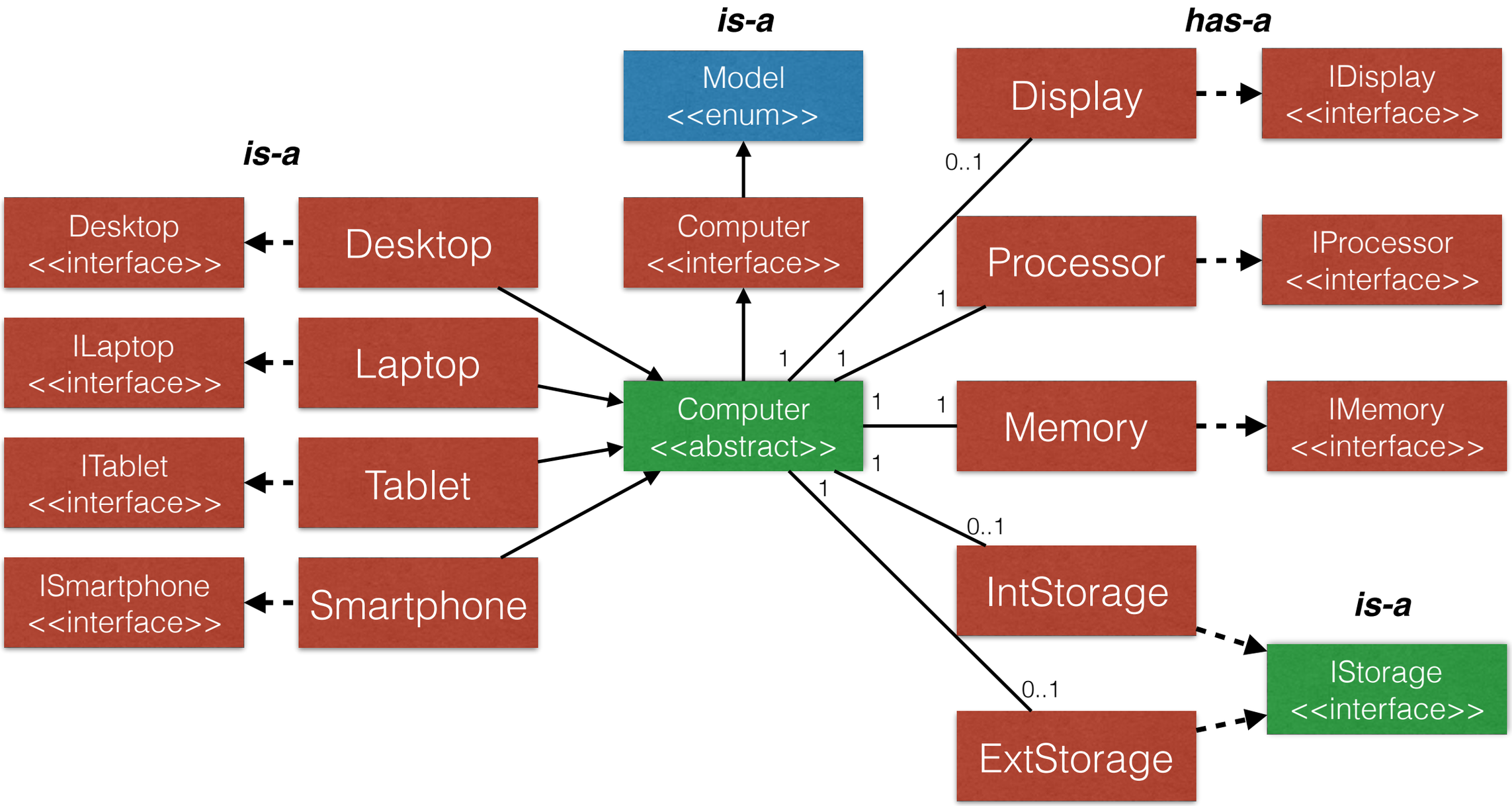


Inheritance-Based Model

- Observations
  - Classes for different types of computers (e.g. Laptop) are very lightweight due to inheritance from the generic Computer class
  - Using Interfaces to enforces standards of behavior on the various types of computers
    - Use of enums improves validation of input data as well as reliability of results
- Pros & Cons

Pros	Cons
Easy to implement common data & behavior via inheritance	Two classes required for each object - one for interface and one for base class
Easy to require behaviors using interfaces	
Unique behaviors are localized in the subclasses they pertain to	Care needed to ensure that generalized services in parent class doesn't become fragmented with services in subclasses.

Composition-Based Model

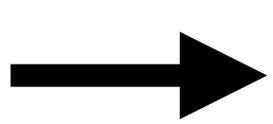


Composition-Based Model

- Observations
  - Changing the Computer class to abstract forced a single change to `main()`

```
switch (computerType) {
          case "desktop":
          computers.add(new Desktop(currJsonComputer));
             break;
          .
          .
          .
          default:
            Scomputers.add(new Computer(currJsonComputer));

      }
```



```
switch (computerType) {
          case "desktop":
          computers.add(new Desktop(currJsonComputer));
             break;
          .
          .
          .
          default:
            System.out.println("Invalid computer type: "+ computerType);
      }
```

- Pros & Cons

Pros	Cons
Same as inheritance-based model	Same as inheritance-based model
Stricter enforcement of specific Computer type	