Software Engineering

CS305, Autumn 2020

Week 7

Class Progress…

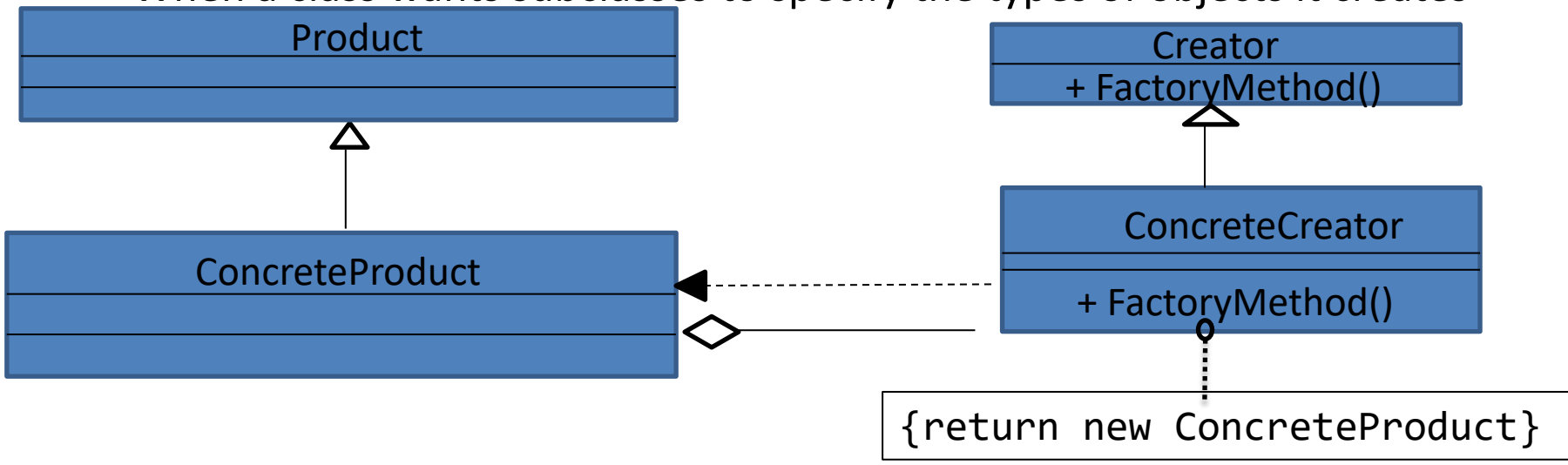
- Last week:
  - Architectural styles
    - Shared services and servers, repository, layered
  - Detailed design
    - Design patterns
    - Singleton

Class Progress…

- This week:
  - Design patterns, Design principles, Rational Unified Process

Factory Method Pattern

- Intent: define an interface for creating an object, and let applications decide which object type to create.
- Applicability
  - When the exact type of object to be created is known at runtime
  - When a class needs control over object creation
  - When a class wants subclasses to specify the types of objects it creates



Factory Method Pattern

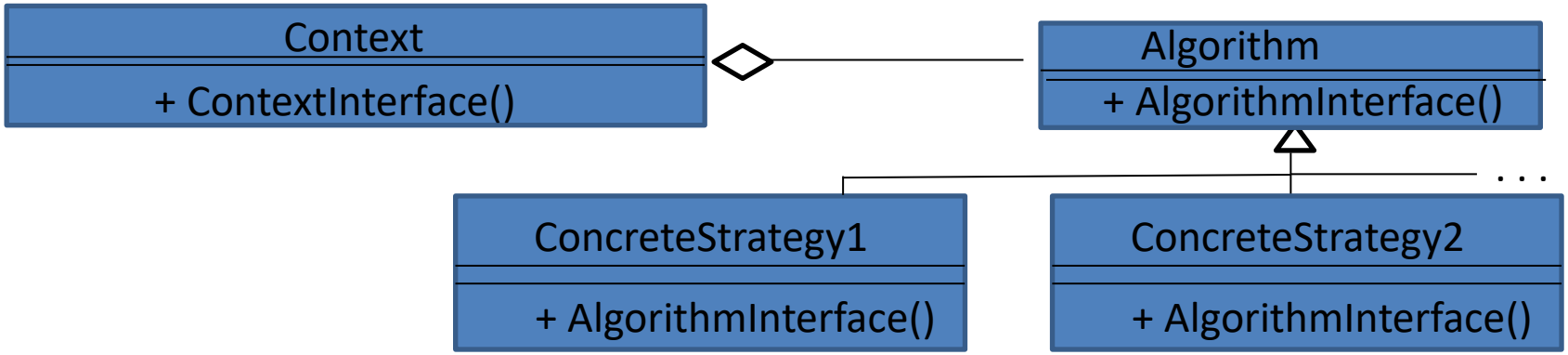
```
Vehicle* VehicleFactory(VehicleType type, Color c) {
        if(type == BUS)
                return new Bus(c);
        else if(type == CAR)
                return new Car(c);
        else
                return NULL;
}

int main() {
        Vehicle* redCar = VehicleFactory(CAR,RED);
        Vehicle* blueBus = VehicleFactory(BUS,BLUE);
}
```

VehicleFactory.cpp, FactoryDemo.java

Strategy Pattern

- Intent: encapsulate each one of a family of algorithms in a separate class and make their usage agnostic
- Applicability
  - When the exact type of object to be created is known at runtime
  - When a class needs control over object creation
  - When a class wants subclasses to specify the types of objects it creates



Some Commonly Used Patterns

- **Visitor** – separate the algorithm from the data structure on which it operates e.g. finding minimum in a binary tree, finding maximum in a binary tree, finding multiples of a given number in a binary tree.
- **Observer** - notify dependents when object changes
- **Iterator** – access elements of a collection without knowing about underlying representation
- **Proxy** – a surrogate controls access to an object

Choosing a Pattern

- Broad guidelines
  - Understand design context
  - Examine the patterns catalogue
  - Identify and study related patterns
  - Apply suitable pattern
- Avoid:
  - Overusing patterns