Design Patterns (2)

Categories of Patterns

- Creational Patterns
  - We already saw Factory and Singleton
- Structural Patterns
  - Already saw Adapter
- Behavioral Patterns
  - Already saw Observer

Today, More Patterns

- Goal: extend your repertoire with commonly-used patterns

Builder

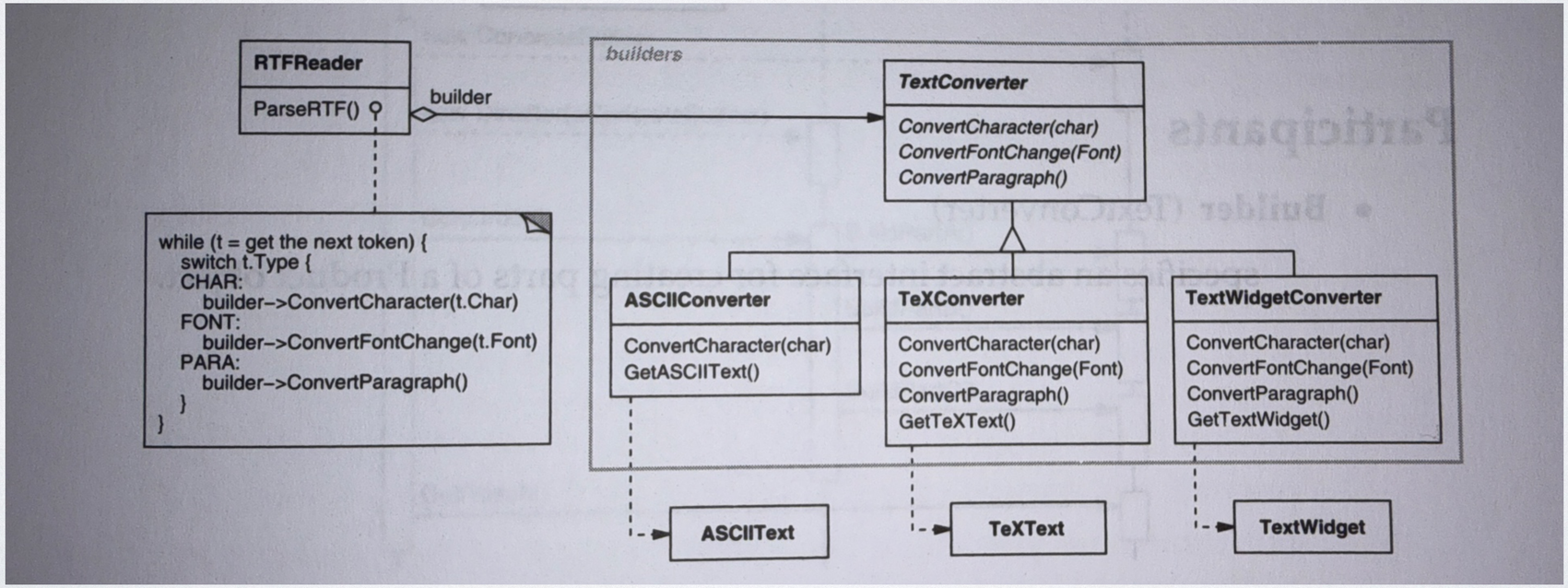
- Why initialize something *all at once* when you can do it in *stages?*
  - (Well, atomicity is nice when you can have it)
- Sometimes an object is composed of complicated parts
  - Want to separate part construction from part assembly
- Sometimes a different representation is needed during construction time
  - A cake goes on a plate, but batter goes in a bowl

Participants

- Builder
    - Specifies an abstract interface for creating parts
- ConcreteBuilder
        - Implements Builder
- Director
    - Constructs the object
- Product
    - Represents the object under construction

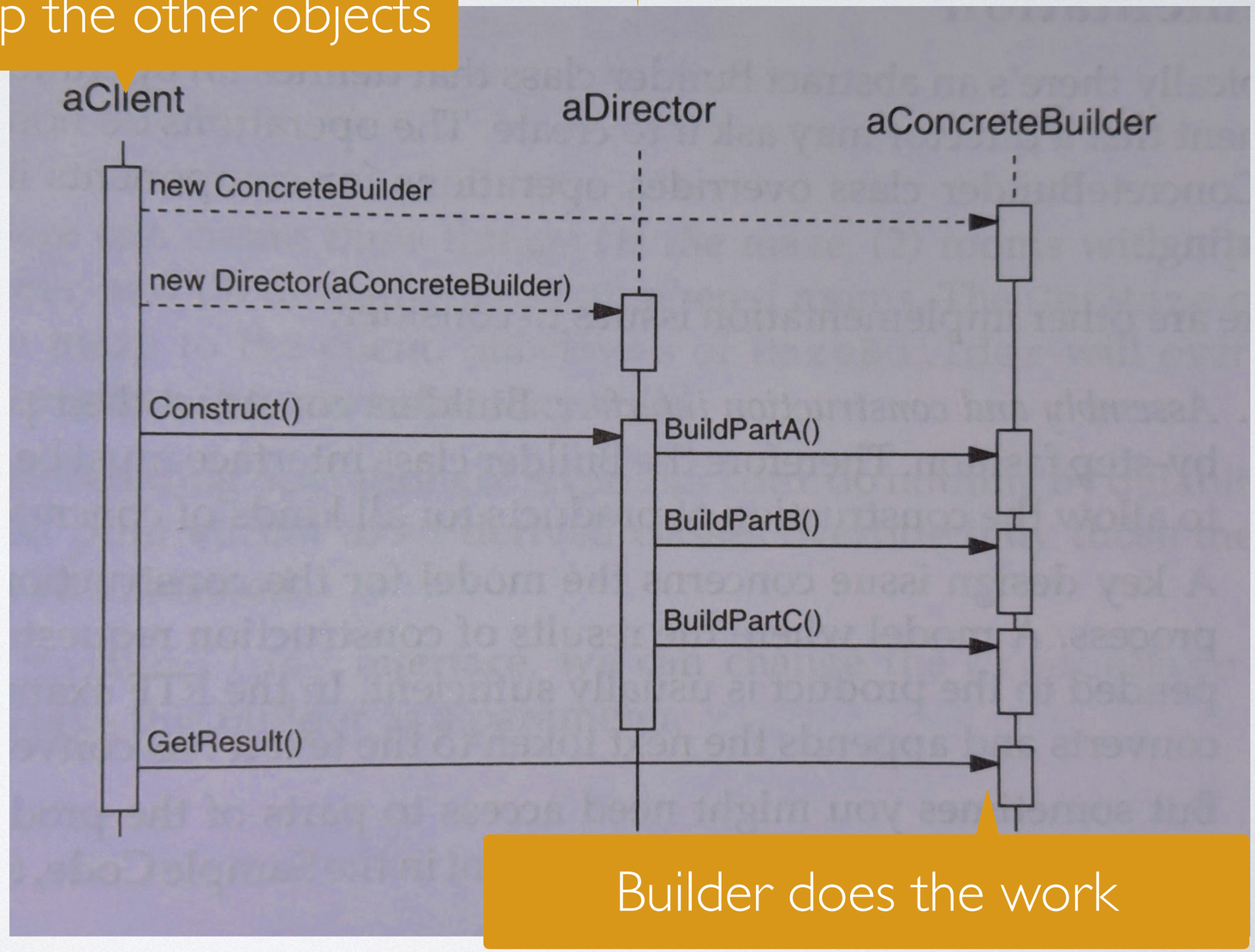
Example

- RTFReader can convert to several different formats while reading. TextConverter is a builder with several subclasses.



Interaction Diagram

- Each "swimlane" represents the actions by one object
- Rectangles represent one invocation (call/return)



Example Product

```
/**
 * Car is a product class.
 */
public class Car {
    private final CarType carType;
    private final int seats;
    private final Engine engine;
    private final Transmission transmission;
    private final TripComputer tripComputer;
    private final GPSNavigator gpsNavigator;
    private double fuel = 0;

    public Car(CarType carType, int seats, Engine engine, Transmission transmission,
               TripComputer tripComputer, GPSNavigator gpsNavigator) {
        this.carType = carType;
        this.seats = seats;
        this.engine = engine;
        this.transmission = transmission;
        this.tripComputer = tripComputer;
        if (this.tripComputer != null) {
            this.tripComputer.setCar(this);
        }
        this.gpsNavigator = gpsNavigator;
    }
```

Credit: https://refactoring.guru/design-patterns/builder/java/example

Example Builder Interface

```
/**
 * Builder interface defines all possible ways to configure a product.
 */
public interface Builder {
    void setCarType(CarType type);
    void setSeats(int seats);
    void setEngine(Engine engine);
    void setTransmission(Transmission transmission);
    void setTripComputer(TripComputer tripComputer);
    void setGPSNavigator(GPSNavigator gpsNavigator);
}
```

Example Concrete Builder

```
**
 * Concrete builders implement steps defined in the common interface.
 */
public class CarBuilder implements Builder {
    private CarType type;
    private int seats;
    private Engine engine;
    private Transmission transmission;
    private TripComputer tripComputer;
    private GPSNavigator gpsNavigator;

    public void setCarType(CarType type) {
        this.type = type;
    }

    @Override
    public void setSeats(int seats) {
        this.seats = seats;
    }

    @Override
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    @Override
    public void setTransmission(Transmission transmission) {
```

Facade

facade /fəˈsäd/ (also façade)
noun

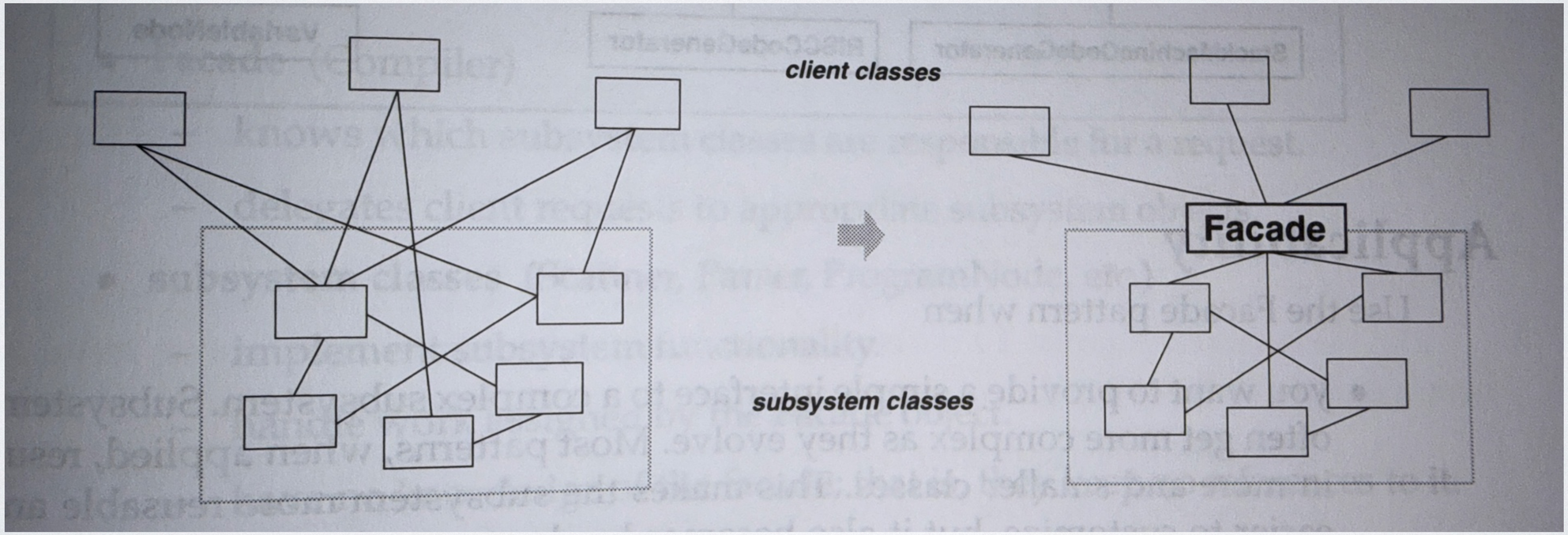
the face of a building, especially the principal front that looks onto a street or open space.

- an outward appearance that is maintained to conceal a less pleasant or creditable reality: *her flawless public facade masked private despair*.

(Oxford English Dictionary)

Facade Conceals Complexity

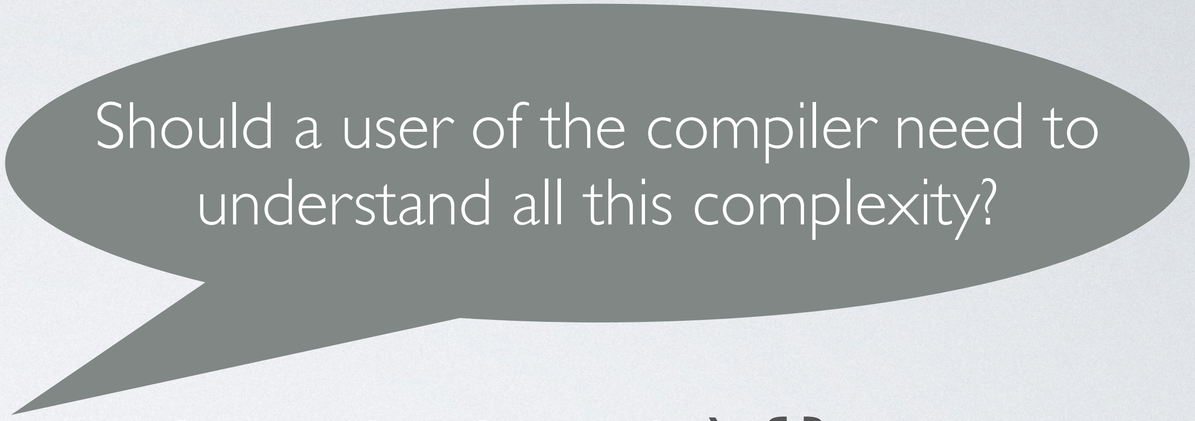
- Offers a simpler interface
- Reduces coupling



Use When

- Want to provide a simple interface for a complex subsystem
  - Note: does *not* provide encapsulation or information hiding
- Want to decouple a subsystem from clients
- Want to add layers to an existing system (a facade can define each layer)

Example: Compiler




```
class Scanner {}
class Parser {
  void parse (Scanner s, NodeBuilder b){}
}
class NodeBuilder {…}
class Node {…}
class CodeGenerator {…}
```

Compiler

```
class Compiler {
  void compile {
    Scanner s = …
    NodeBuilder b = …
    Parser p = …
    p.parse(s, b);
    CodeGenerator g = …
    parseTree = b.getRootNode();
    parseTree.traverse(g);
  }
}
```

Now, anyone can use the compiler much more easily.

Compiler is a *facade*.

Are These So Hard?

- For the rest of class, work with a partner:

  1. Read about one pattern in the Gang of Four book (free online from the library)

  2. Briefly summarize the pattern on Gradescope for today's ICA.

  3. Explain the pattern to your partner. Summarize their pattern too.