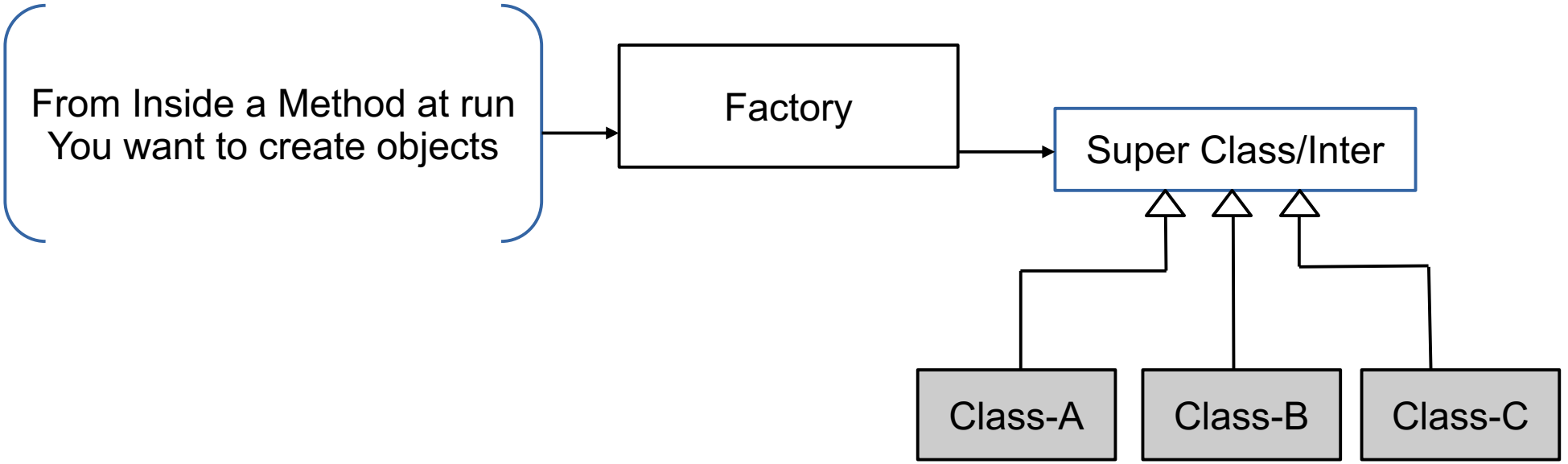
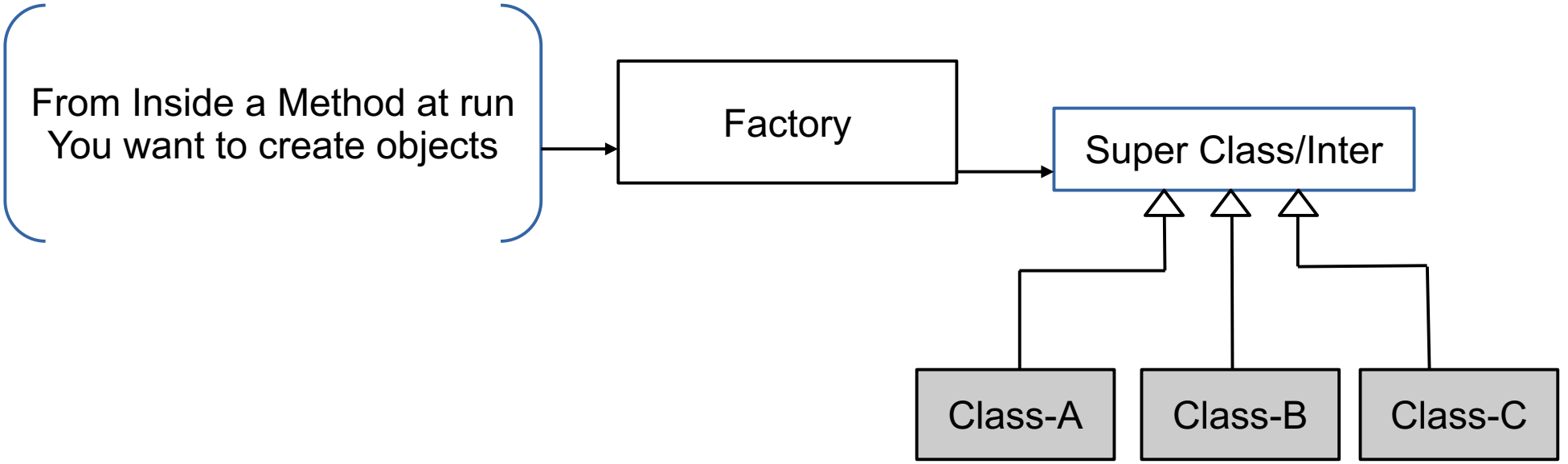
Factory Method Pattern

Problem – **Factory Method**



Problem – Factory Method

- Sometimes one of our application classes only knows **when a new object** should be created, but **not which specific sub-type** of that object should be created.
- In some cases, instantiation is complex and you might need to setup do some settings/configurations to create objects.
- You want to have the creation of objects flexible.

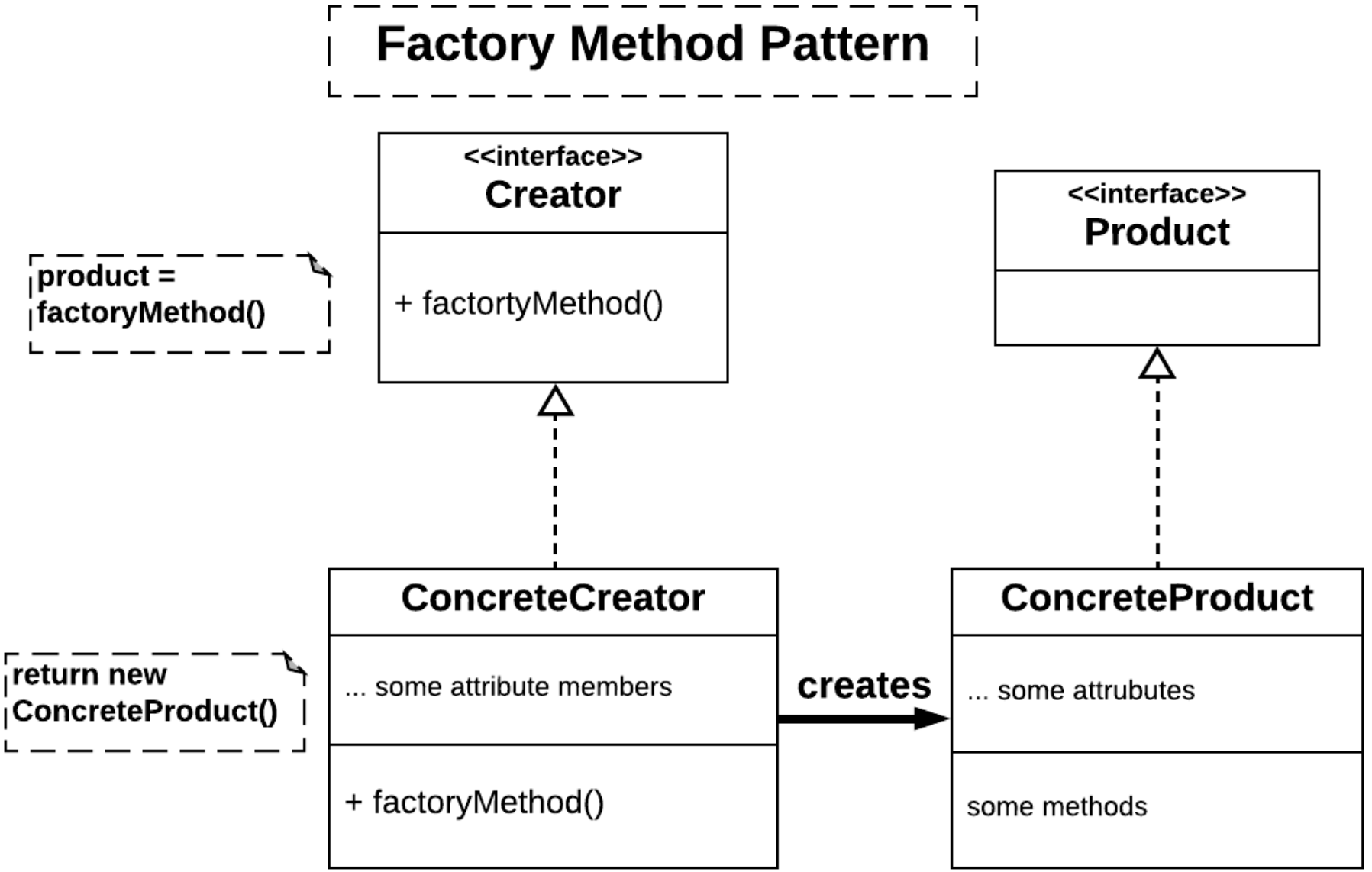


Factory Method

- Definition from Book: "*Factory method defines* ***an interface for creating an object****, but* ***let subclasses decide*** *which class to instantiate. Factory Method lets a class defer instantiation to subclasses.*"

- Factory pattern encapsulates object creation and delegates the responsibility to another class.

Factory Method Pattern



Bank Account Example

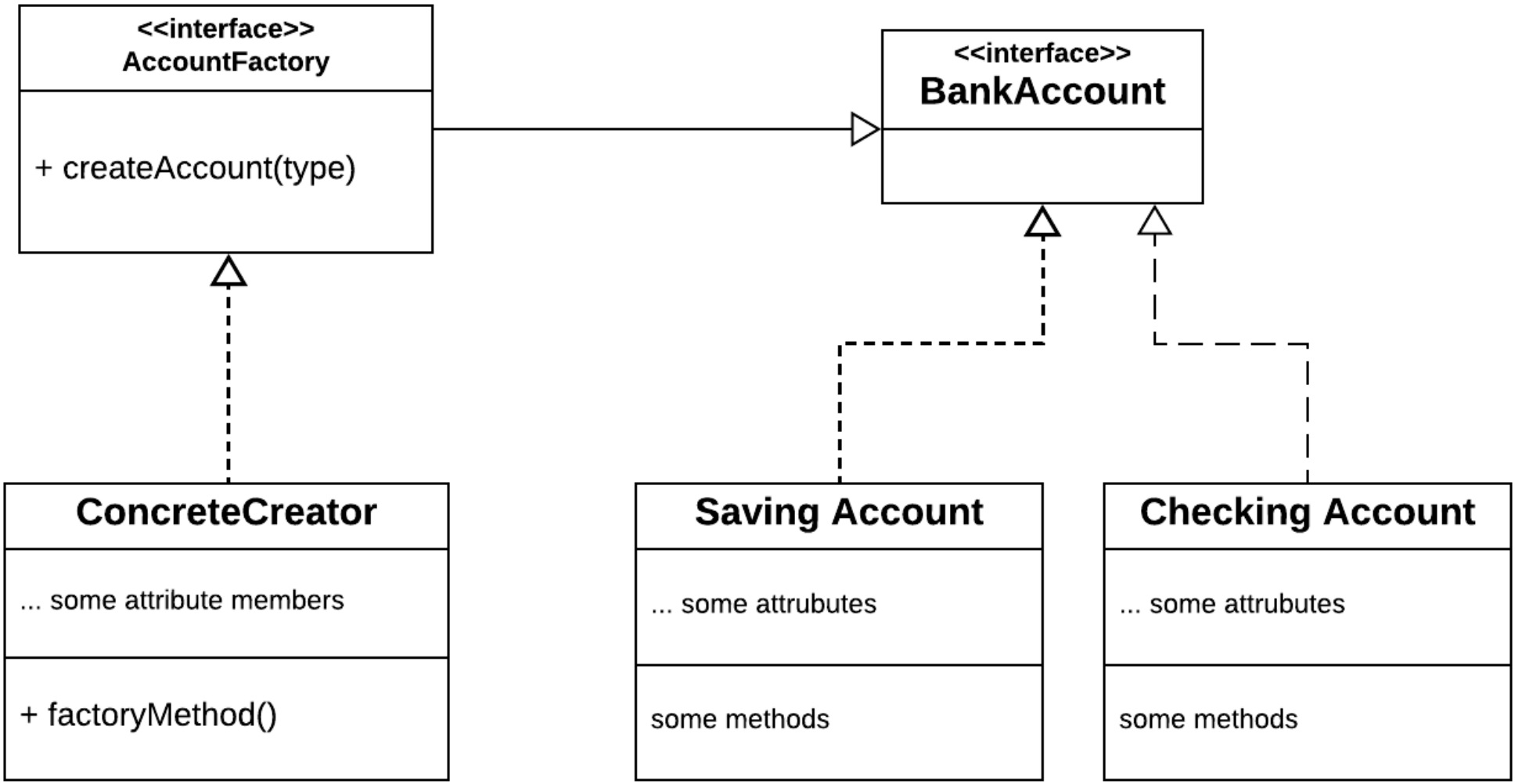
- Creating Bank accounts may include multiple bank internal sub-processes that we want to hide.

- Bank front desk employees should not deal with all of the internal subprocesses, they should be able to simply create accounts for customers.

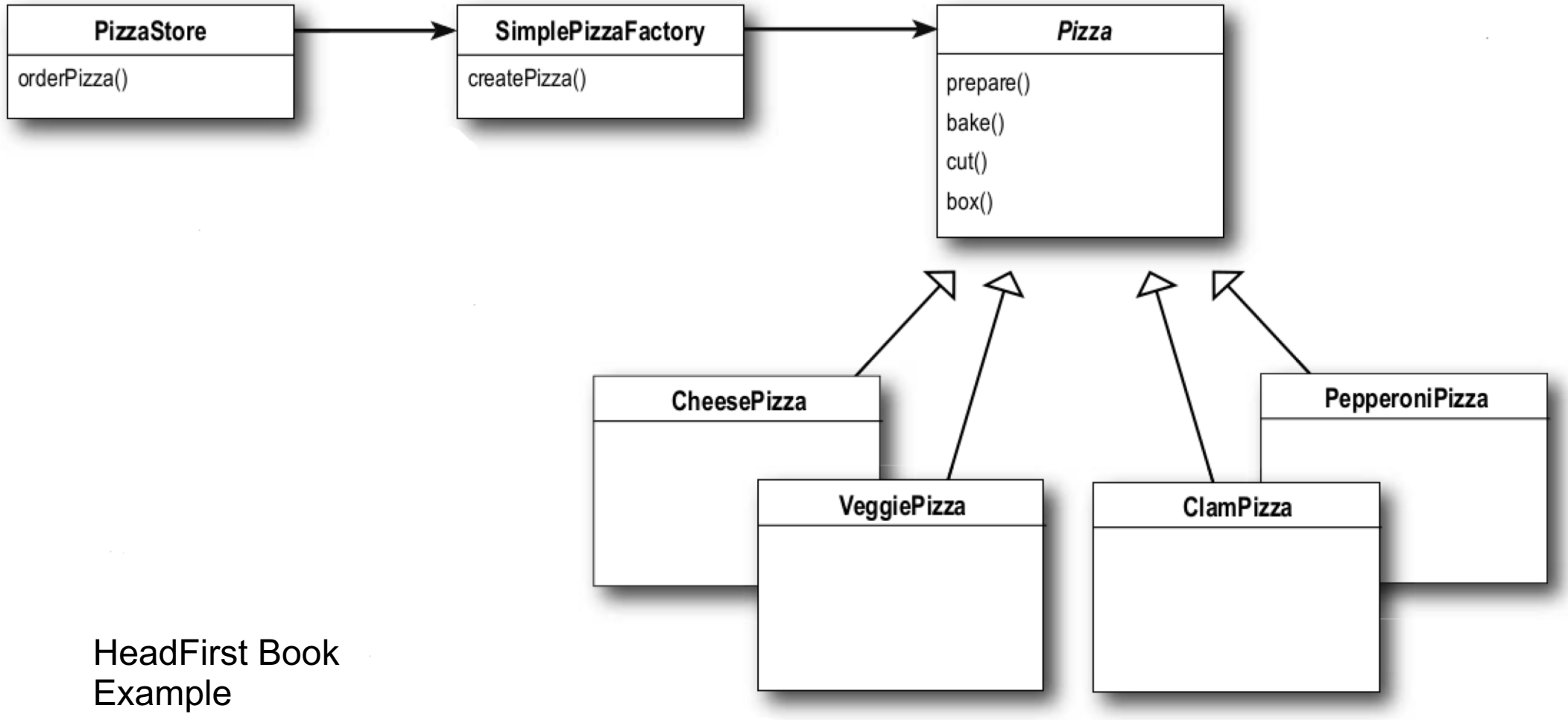
- A front desk person just ask the customer which kind of bank accounts (checking or saving) accounts she/he wants to have and accounts can be easily created.

- BankAccount bankAccount= accountFactory().createAccount()

Example : Creating Bank Accounts



Example: Pizza Store with different kind of Pizza



HeadFirst Book Example

Participants

- **Product**
  - defines the interface of objects the factory method creates.
- **ConcreteProduct**
  - implements the Product interface.
- **Creator**
  - It declares the factory method, which returns an object of type Product. Creator may also define a default implementation of the factory method that returns a default ConcreteProduct object.
  - It may call the factory method to create a Product object.
- **ConcreteCreator**
  - overrides the factory method to return an instance of a ConcreteProduct.

When use the Factory Method Pattern

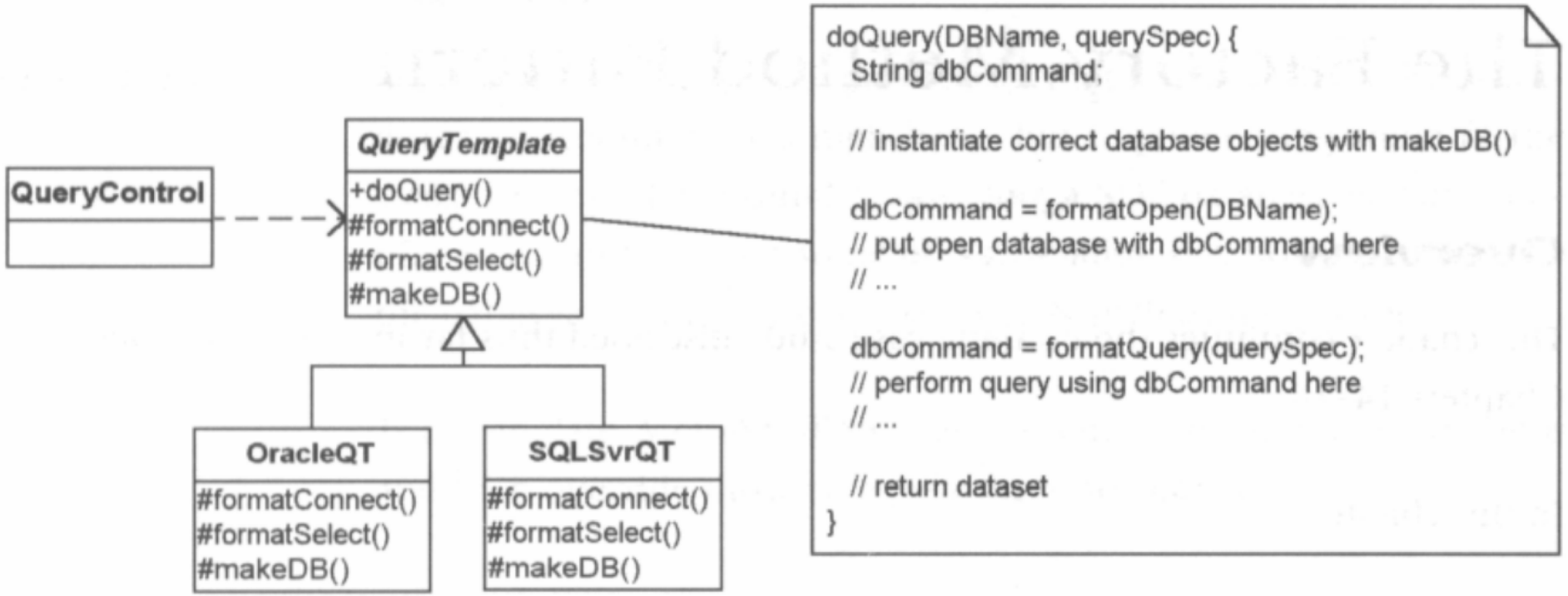
- Use Factory Method when
  - a class **can't anticipate** the class of objects it must create.
  - a class wants its **subclasses to specify the objects it creates**.
  - classes **delegate responsibility** to one of several helper subclasses, and you want to **localize the knowledge** of which helper subclass is the delegate.

Consequences

- **Provides hooks for subclasses.**
  - Creating objects using factory method is more flexible than creating an object directly.
  - Factory Method gives subclasses a hook for providing an extended version of an object.
- **Connects parallel class hierarchies.**
  - In our examples we've considered that the factory method is only called by Creators. But this doesn't have to be the case; clients can find factory methods useful, **especially in the case of parallel class hierarchies**.

Example – 3 : Implement this on your own

- The method makeDB instantiates the appropriate database object.
- Query-Template does not know which database object to instantiate. It only knows that one must be instantiated and provides an interface for its instantiation.
- The derived classes from Query-Template will be responsible for knowing which ones to instantiate.



Example from GangOfFour Book