Creational Patterns

"Code with Passion!"

Creational Patterns

Patterns that are concerned with object instantiation

- Factory patterns
  - Simple factory
  - Factory method pattern
  - Abstract factory pattern
- Singleton pattern
- Builder pattern
- Prototype pattern
- Object pool pattern

Creational Patterns

Patterns that are concerned with object instantiation

- Factory patterns
  - Simple factory
  - Factory method pattern
  - Abstract factory pattern
- Singleton pattern
- Builder pattern
- Prototype pattern
- Object pool pattern

Factory Patterns

Three different types of factory patterns

- Simple factory
  - Not really a design pattern
  - Rather, it is a simple encapsulation of object creation into another class
- Factory method pattern (Factory pattern)
  - Defines an abstract method called “factory method” in abstract class
  - This abstract factory method is implemented by concrete factory subclasses – the subclasses are responsible for object creation
- Abstract factory pattern
  - Provides a way to encapsulate a group of individual factories that have a common theme without specifying their concrete classes

Simple Factory

Simple Factory

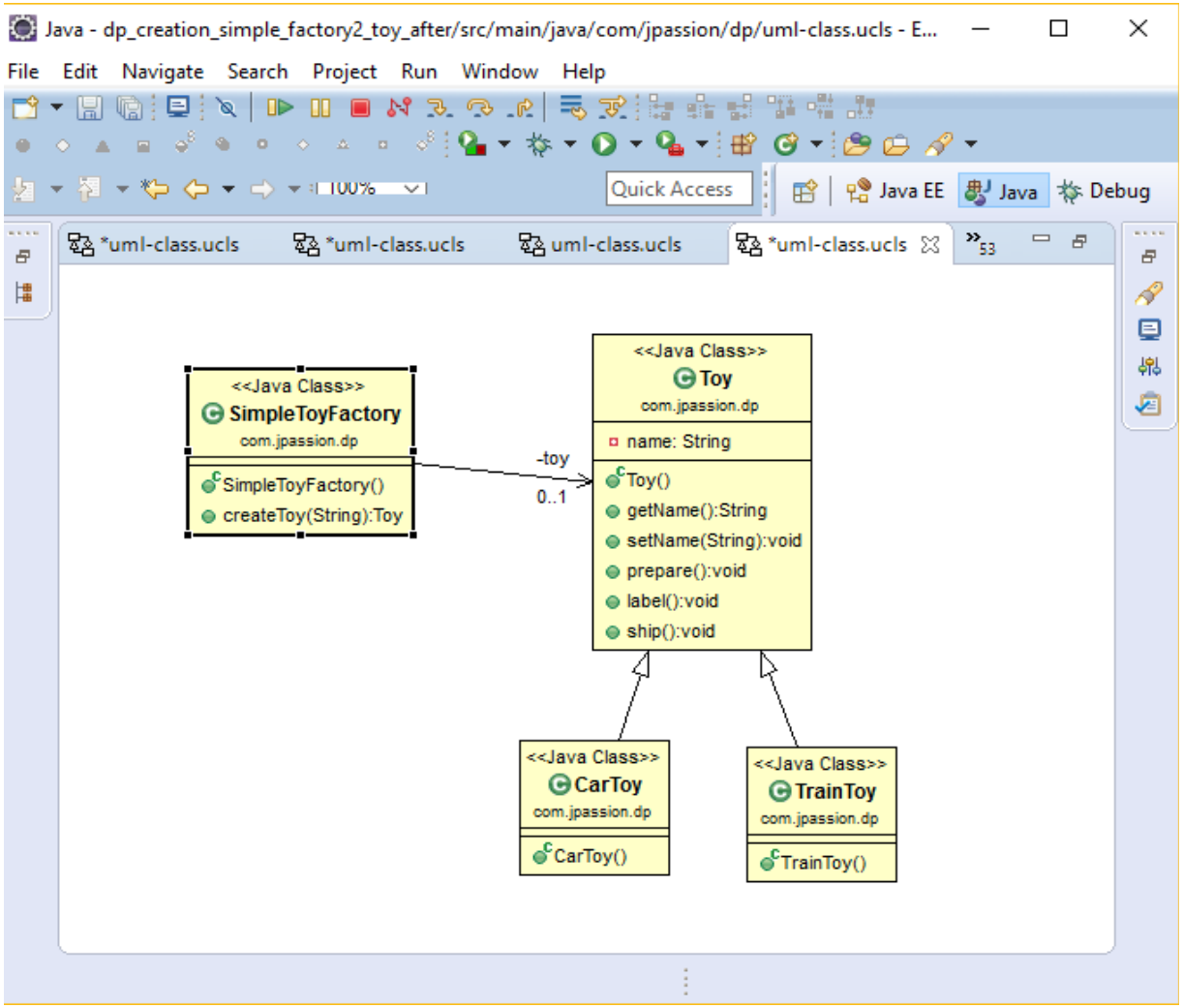
- What is it?
  - Not really a design pattern
  - Rather, it is a simple encapsulation of object creation into another class
- When to use it?
  - When you need to decouple “object creation code” from “business logic code”
  - When object creation logic changes often – encapsulate what changes
  - When object creation logic needs to be reused in other part of the application
- Downside of Simple Factory
  - When new type of object needs to be created, the “object creation code” needs to be modified – violates Open-Close principle

Example Scenario of Simple Factory

- We are a toy company and we want to encapsulate creating different kinds of toy into a *ToyFactory* class
  - *CarToy*, *TrainToy*

- Future changes
  - We want to create new kind of toy such as *PlaneToy* modifying the *ToyFactory* class

Simple Factory



Lab:

Exercise 1: Simple Factory
9001_dp_creation.zip

Factory Method Pattern (Factory Pattern)

Factory Method Pattern

- What is it?
  - Defines an abstract method called “Factory method” in abstract class
  - This abstract factory method is implemented by factory subclasses – the factory subclasses are responsible for object creation
  - A factory subclass can be chosen at runtime - “Factory Method lets a class defer instantiation to subclasses” - addresses the downside of the Simple Factory

- When to use it?
  - When you don't know ahead of time what class object you need to create – it will be determined during runtime via choosing concrete factory subclass during runtime

Example Scenario of Factory Method Pattern

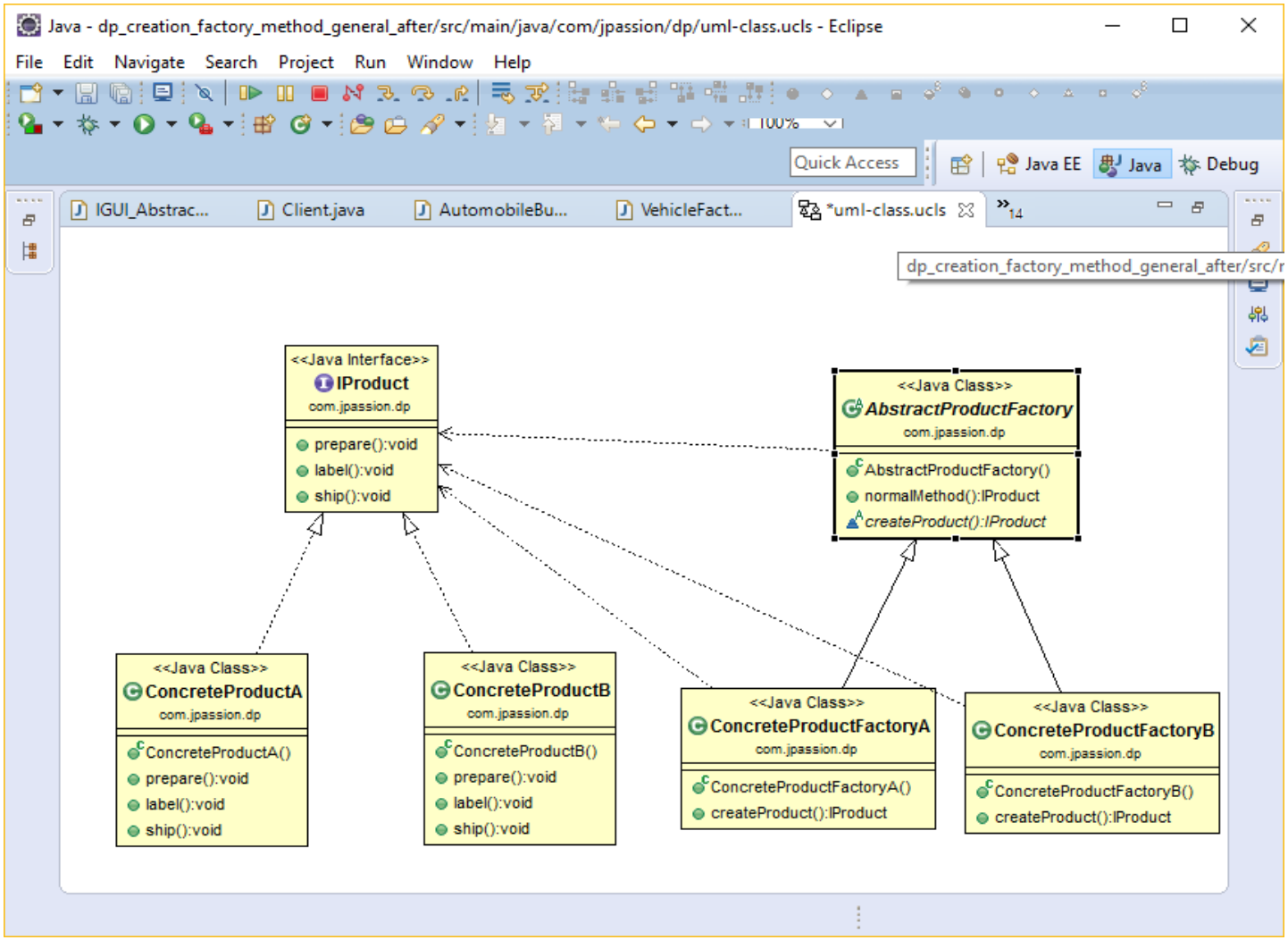
- We are toy company and we want to create different kinds of toys using different toy factory classes during runtime

- Changes in the future
  - Creating a new toy by adding a new factory class

Participants

- AbstractProductFactory (abstract class)
  - Contains abstract factory method for object creation
- ConcreteProductFactory classes
  - Implements the abstract factory method of the AbstractProductFactory abstract class
  - Create concrete product
- Product
  - Declares an interface for a type of product objects
- ConcreteProduct
  - Defines a product to be created by ConcreteProductFactory classes
- Client
  - Uses the interfaces declared by the AbstractProductFactory interface and Product

Factory Method Pattern



Lab:

Exercise 2: Factory Method Pattern 9001_dp_creation.zip

Abstract Factory Pattern

Abstract Factory Pattern

- Provides an AbstractFactory interface for creating families of related objects without specifying their concrete classes
  - Provides a way to encapsulate a group of individual factories
- By writing code that uses the AbstractFactory interface, we decouple our code from actual factory that creates the products
- Because our code is decoupled from the actual products, we can substitute different factories to get different behaviors

Example Scenario #1 of Abstract Factory

- We are a manufacturer of computers and we want to create multiple brands based on price range
  - High-end computer
  - Middle-end computer
  - Low-end computer
- Each computer is made of multiple parts
  - Monitor
  - Processor
  - HDD (Hard disk storage)
- High-end computer is made of high-end Monitor, Processor, HDD
- Future changes
  - We might want to create Super-high-end computer

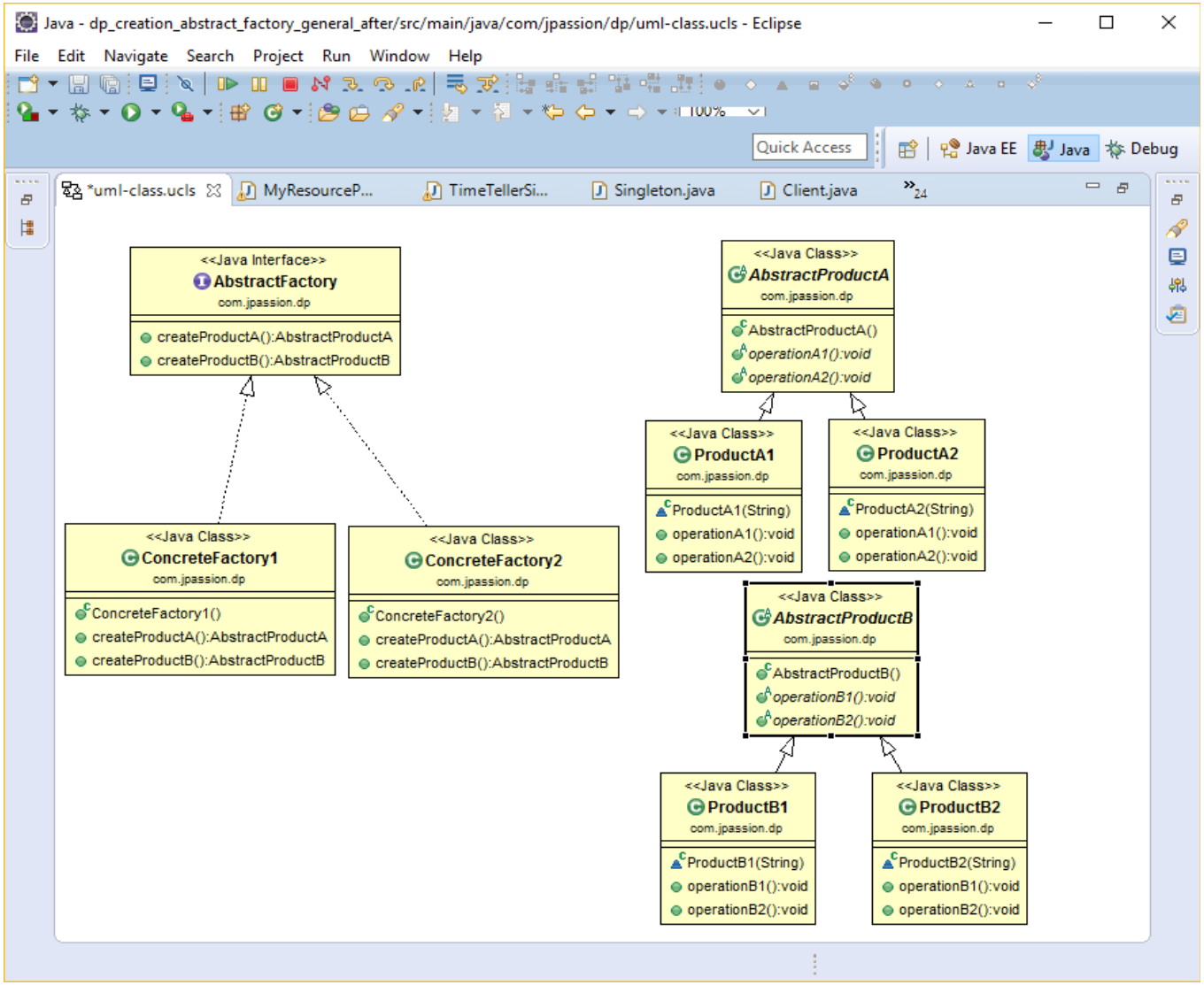
Example Scenario #2 of Abstract Factory

- We are a software vendor of GUI libraries and we want to create them for each OS
  - Windows
  - Mac OS
- Each GUI library has multiple GUI components
  - Menu
  - TitleBar
  - Frame
- Future changes
  - We might want to create one for Linux

Participants

- AbstractFactory
  - Declares a interface for operations that create a family of abstract products.
- ConcreteFactory
  - Implements operations to create concrete family of products.
- AbstractProduct
  - Declares an interface for a type of product objects
- Product
  - Defines a product to be created by the corresponding ConcreteFactory; it implements the AbstractProduct interface.
- Client
  - Uses the interfaces declared by the AbstractFactory interface and AbstractProduct classes.

Abstract Factory Method Pattern



Lab:

Exercise 3: Abstract Factory Pattern
9001_dp_creation.zip

Singleton Pattern

Singleton Pattern

- What is it?
  - Ensure a class has only one instance, and provide a global point of access to it
- When to use it?
  - Use it for centralized management of a single resource

Example Scenarios

- Window manager object
- File system object

Participants

- Singleton class
  - Has a private constructor – prevents other objects from creating a new object via “new” keyword
  - Has a static method that returns the single instance

Lab:

Exercise 4: Singleton Pattern
9001_dp_creation.zip

Builder Pattern

Builder Pattern

- What is it?
  - > Separates the construction of a complex object from its representation so that the same construction process can create different representations
  - > Allows finer control over the construction process. - unlike other creational patterns that construct products in one shot, Builder pattern constructs the product step by step under the control of “director”
- When to use it?
  - > Use it to simplify object creation when object needs to be created with complex set of initial arguments
  - > Use it when an object creation need to be done flexibly

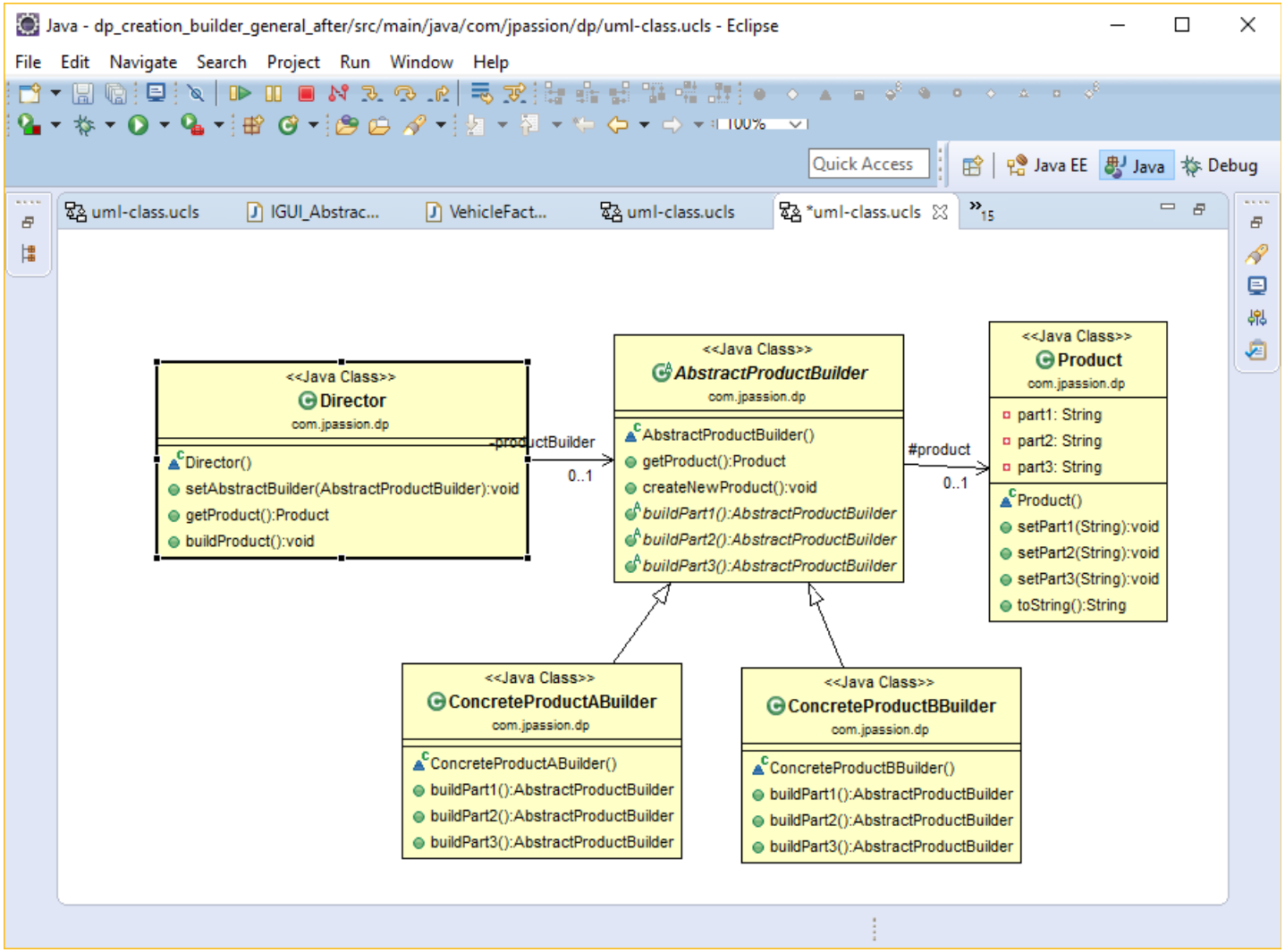
Example Scenario of Builder Pattern

- Car has many different parts and we want to build the car step by step abstracting away the internal representation
- Future changes
  - The internal parts of a Car might change

Participants

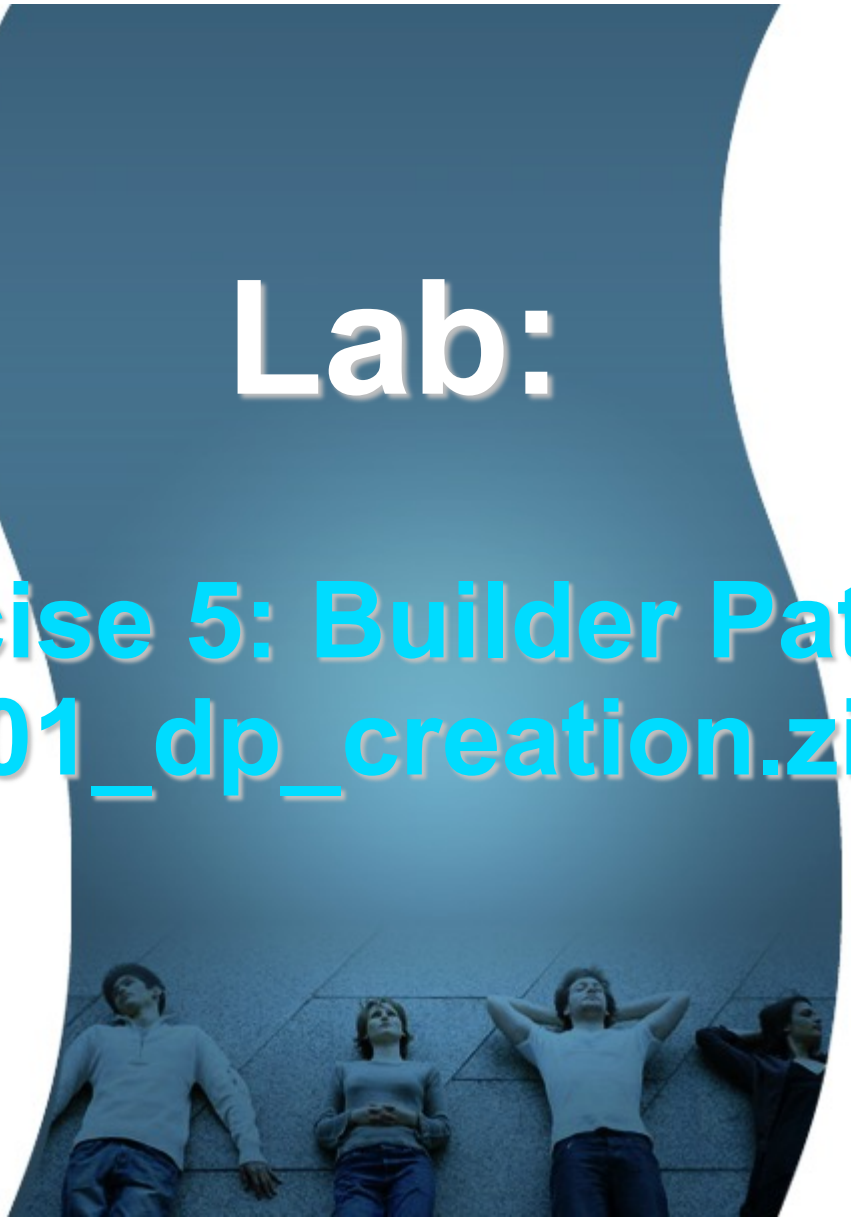
- AbstractBuilder
  - Abstract interface for creating parts of a Product object
- ConcreteBuilder
  - Constructs and puts together parts of the product by implementing the AbstractBuilder interface
  - Defines and keeps track of the representation it creates and provides an interface for saving the product.
- Director
  - Constructs the complex object using the AbstractBuilder interface.
- Product
  - Represents the complex object that is being built

Builder Pattern



Lab:

Exercise 5: Builder Pattern
9001_dp_creation.zip



Prototype Pattern

Prototype Pattern

- What is it?
  - Allows to create a new object by cloning
- When to use it?
  - Use it when creating a new object is expensive

Example Scenario

- Creating and configuring data connection object is very expensive

Participants

- Client
  - Creates a new object by asking a prototype to clone itself.
- Prototype interface
  - Declares an interface for cloning itself
- ConcretePrototype
  - Implements the operation for cloning itself.

Lab:

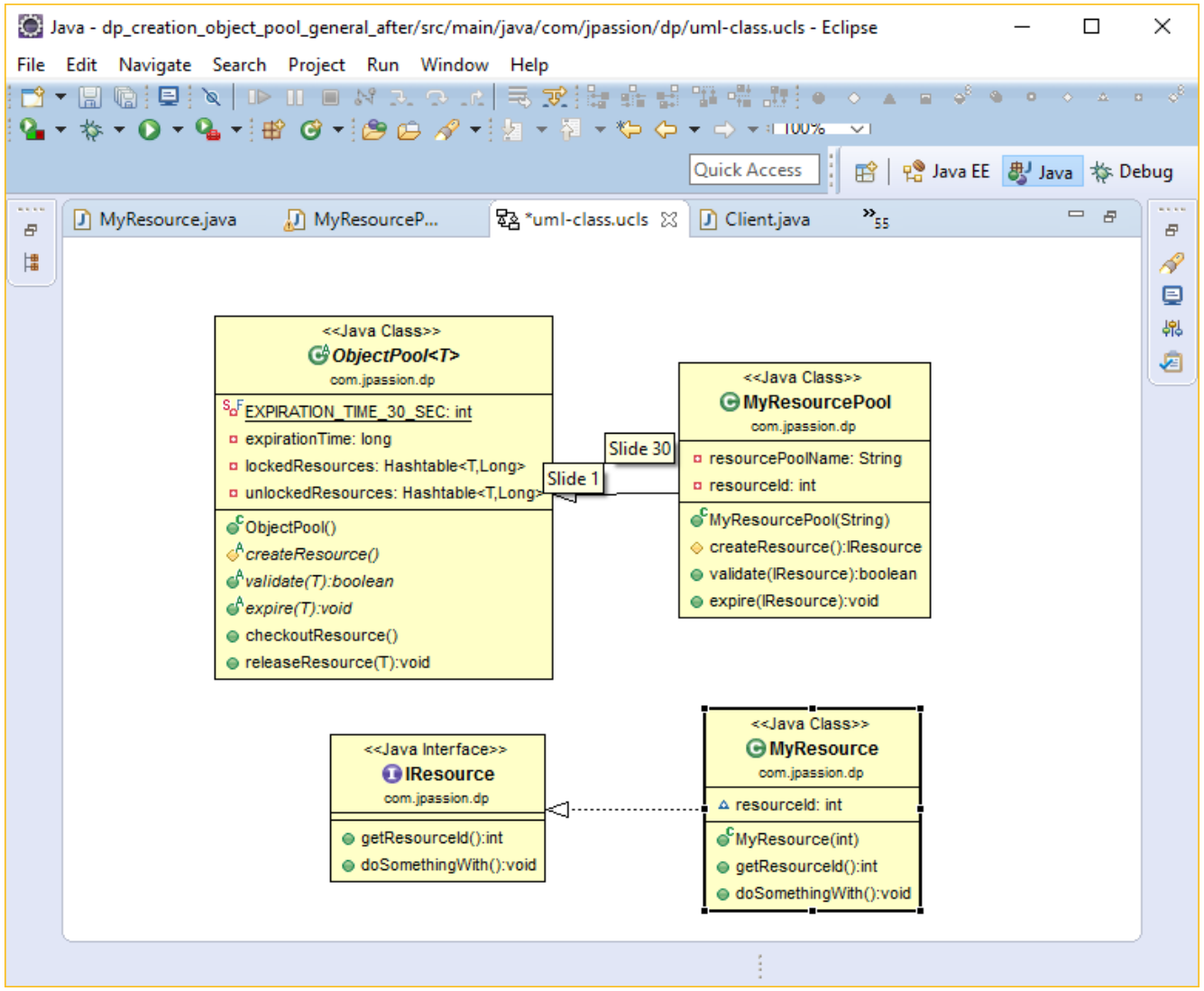
Exercise 6: Prototype Pattern 9001_dp_creation.zip

Object Pool Pattern

Object Pool Pattern

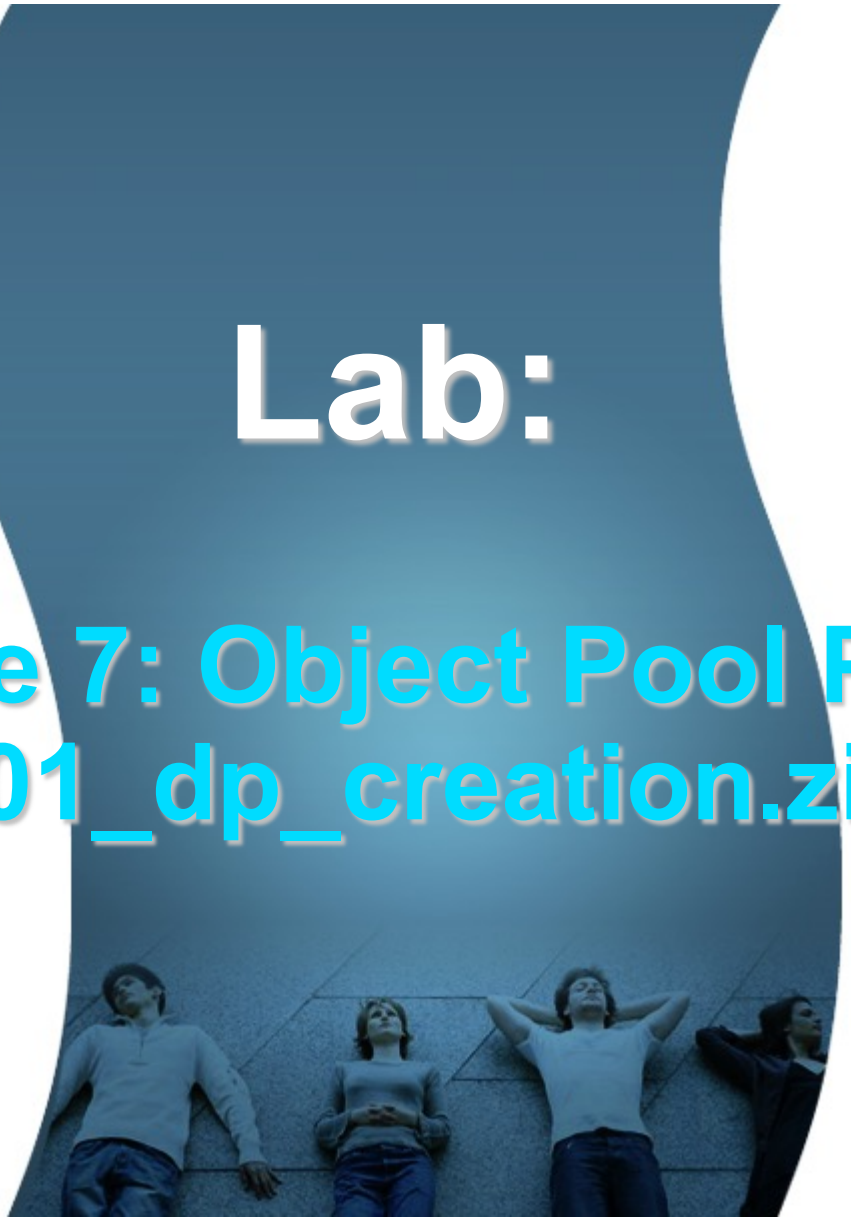
- What is it?
  - Reuse and share objects that are expensive to create
- Examples
  - Database connection pool
  - Thread pool

Object Pool Pattern



Lab:

Exercise 7: Object Pool Pattern
9001_dp_creation.zip



Code with Passion!