CISC 3120

# C15: Dependencies, Observables, Properties, and Concurrency

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

# Outline

- Recap and issues
- Important problems
  - Share data among components
  - Design responsive user interface
- Dealing with dependency
  - Passing data and objects among UI components
- Observer pattern
- Bindings and properties
- Concurrencies in JavaFX

# Dependency

- An object depends on outside values (data)
  - An object (the client) depends on the states and behaviors of another object (the server)
- Example scenarios or applications?
- How do we handle it elegantly?
  - Objective
    - Changing the code of the server should not result in the change of the code of client.

# Dependency Injection

- How one object supplies the dependencies of another object
- A few common techniques
  - Not to use dependency injection
  - Use dependency injection
    - Via setter methods
    - Via constructor methods
    - Via inheritance (implementing interface or extending a class)

# Example App: Tracking Student's Grade

- Not to use dependency injection
- Use dependency injection
  - Via setter methods
  - Via constructor methods
  - Via inheritance (implementing interface or extending a class)

# Dependency Injection: Discussion

- Via constructors
  - Can make sure dependencies are instantiated and valid from the outset;
  - But cannot change the dependencies
- Via setters
  - Can change the dependencies flexibly;
  - However, cannot guarantee the setters are called (need to validate dependencies state)
- Via interfaces
  - A variation of the setters method with delayed setter implementation
  - Flexible to delegate setters responsibilities to the dependencies themselves, clients, or other objects

# Frameworks and Loosely Coupled Objects

- The examples are for illustrating the basic concept
- How do we apply it to design applications with loosely couple objects?
- Frameworks and containers
  - Often comes with assemblers that instantiates and wires the objects together
- You may do it yourself!
  - Just to make sure to follow the principle

# Working with JavaFX

- Adding "dependency"
  - Not to use dependency injection
  - Use dependency injection
    - Via setter methods
      - Via Node's setUserData method
      - Via Node's getProperties method (what is a property? See properties & bindings)
    - Via constructor methods
    - Via inheritance (implementing interface or extending a class)
- For User Interfaces, consider to create "View" classes

# Questions?

- Concept of dependencies of objects
- Concept of dependency injections
- Dependency injection mechanisms
- Example applications
  - In the Sample Programs repository

# Changes and Dependencies

- When making changes to objects, make changes to the clients (e.g., views) of the objects
  - Dependency injection
  - Pass messages to the clients/invoke clients' methods directly
  - Not always a good solution
    - Why?
- Make changes to objects, and let clients update themselves
  - A good solution
  - But, how do the clients know that the objects change?

# Changes and Dependencies

- When making changes to objects, make changes to the clients (e.g., views) of the objects
  - Dependency injection
  - Pass messages to the clients/invoke clients' methods directly
  - Not always a good solution, sometimes not even possible
    - Why?
- Make changes to objects, and let clients update themselves
  - A good solution
  - But, how do the clients know that the objects have changed?

# The Observer Pattern

- One depends on other objects
  - When one object changes, others also needs to change.
- Solution
  - An observable object can have one ore observers, and observers can be notified the changes the observable object
    - An observable object is an object on which the clients depend on
      - Often referred to as the "data" or "model"
    - An observer object is the client that depends on the observable object
      - Often referred to as the "view", in particular, in GUI applications
- Commonly used in the User Interface design

# Java Observable and Observer

- Java supports the observer pattern
  - It has the Observable class and the Observer interface
    - java.util.Observable
    - java.util.Observer
  - An Observable object can have one or more "observers"
    - An application (1) the Observable's addObserver method to add an observer to the Observable, and (2) calls the Observable's notifyObservers method
  - Each observer must implements the Observer interface
    - The platform calls the Observer's update method when the observer is notified of a change of the Observable

# Using the Observable and Observer

- Preparing an Observable
  - Extending the Observable class
    - Invokes the Observable's setChanged() and notifyObservers(…) methods
  - Add observers to the Observable
    - Invokes the Observable's addObserver(…) method
- Preparing an Observer
  - Creating an object of the Observable
    - An Observer object typically has one or more Observable objects to observe
  - Implementing the Observer interface
  - Override the Observer's update(…) method

# Questions

- The Observer pattern
  - Commonly seen in User Interface development
- The Java Observable and Observer classes
- Example application
  - In the Sample Programs repository

# Dependency and UI Component

- UI components need to response to changes of depended objects
- What are the solutions?
  - Dependency injection
  - Observables and Observers
- Problem
  - In even-driven GUI applications, how do we update UI components when depended objects change in event-driven fashion?
- The discussion also applies to non-UI components

# JavaFX Properties and Bindings

- JavaFX Properties are JavaFX objects and APIs
  - that realize the Observer pattern
  - that follow event-driven programming paradigm

# JavaFX Properties

- JavaFX properties are observable objects in event-driven programming
- Observable notifies observers via events
- Observer listens to and handles events triggered by the changes of the properties objects

# JavaFX Property

- An interface
  - javafx.beans.property
- Typically, use one of many concrete implementing classes
- A property object can have a list of listeners to listen to two types of events
  - Via ObservableValue's addListener(…) and Observable's addListener(…) methods
  - Two types of listeners
    - ChangeListener
    - InvaliadtionListener

# ChangeListener or InvalidationListener?

- Examine the definition of the interfaces
- "eager evaluation" and "lazy evaluation"?

# Questions

- Concept JavaFX properties
- Relationship with the Observer pattern and dependency injection
- Event handling for JavaFX properties
- Example application
  - In the Sample Programs repository

# JavaFX Bindings

- A Binding calculates a value that depends on one or more sources.
  - The sources are usually called the dependency of a binding.
  - A binding observes its dependencies for changes and updates its value automatically.
- A convenient mechanism
  - to express direct (dependency) relationships between objects
  - to define how changes made to one object is "automatically" reflected in another object
- Realized using JavaFX Properties, Observables, ObservableValues
- High-level binding API (Simple bindings)
- Low-level binding API

# Simple Bindings

- Property objects has methods that "bind" a property to another
  - Also referred to Fluent API
    - Fluent: using method chaining, the method calls resemble a "prose" in a natural language.
  - Unidirectional binding, bidirectional binding
- Bindings (java.beans.binding.Bindings) has static factory methods that create simple bindings
- What they do?
  - Create a binding between sources
  - Bind the binding to a property (that serves as an observer to the binding)

# Low-Level Binding APIs

- Generally, extending one of the Binding classes
  - javafx.beans.binding
    - BooleanBinding, DoubleBinding, FloatBinding, IntegerBinding, ListBinding, LongBinding, MapBinding, ObjectBinding, SetBinding, StringBinding
  - Call the superclass's bind(…) method in the subclass's constructor
  - Override the computerValue(…) method
  - See the description and examples in the DoubleBinding class
- Note:
  - All bindings in the JavaFX runtime are calculated lazily.
    - Calling binding's get() method results the calculation of the binding

# Questions

- Concept of bindings
- Bindings in JavaFX components
- Example application
  - In the Sample Programs repository

# Concurrency and JavaFX

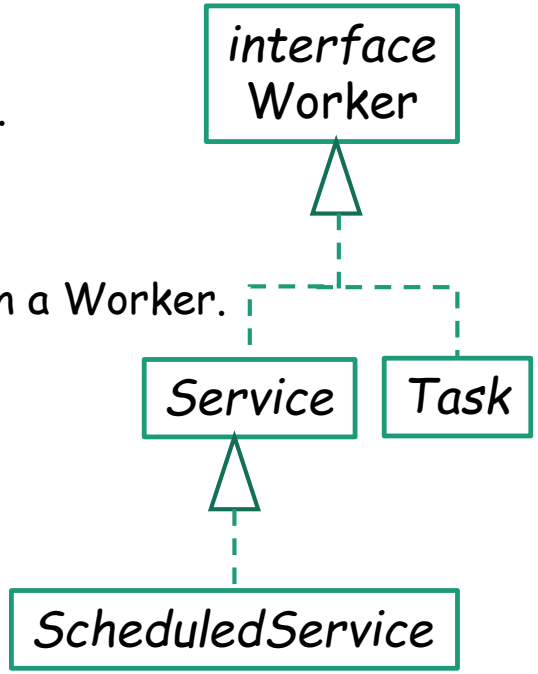
- Concurrency
  - Two or more methods are running simultaneously
    - Each is often referred to as a "thread".
- JavaFX platform runs a "method" to manage UI components, such as, a scene graph
  - JavaFX "UI thread", or JavaFX "Application thread"
  - Scene graphs are accessed and modified sequentially in the "Application thread"
  - Time consuming methods can make UI nonresponsive
- Design for responsiveness
  - delegating time-consuming method (task) execution to background threads
  - utilizing the javafx.concurrent package

# Motivational Examples

- Design consideration
  - event handlers should return quickly, since they are invoked in an event loop
- What if we have lots of work to do when an event occurs?
- Two applications
  - Implementing a Monte Carlo simulation to estimate $\pi$.
    - The simulation can take a while to run.

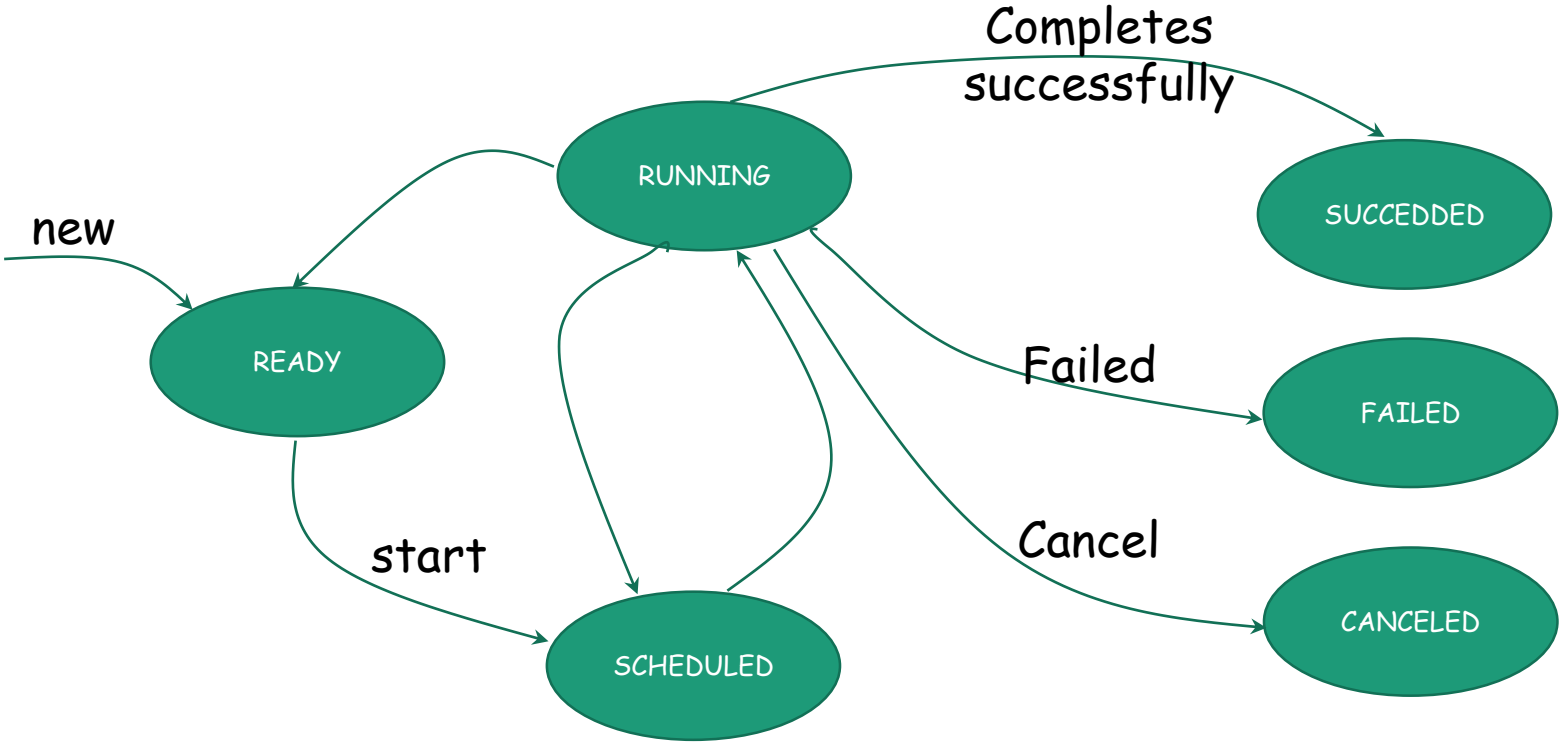
# The Worker Interface

- Worker
  - An object which performs some work in one or more background threads
  - It is an observable object
  - The observers can be the JavaFX Application thread.
- WorkStateEvent
  - An event which occurs whenever the state changes on a Worker.



# Worker States

- Worker.State



# Worker Progress

- Work has three different properties, totalWork, workDone, and progress.
- User's implementation sets the values of the properties
- User's implementation should ensure to stop processing when the worker is canceled

# Task and Service

- Three classes:
  - javafx.concurrent.Task, javafx.concurrent.Service, javafx.concurrent.ScheduledService
- An instance of Task is a one-short worker
  - Create once, run once, and cannot be reused
- A Service creates and manages a Task that performs the work on the background thread.
- A ScheduledService is a Service
  - which will automatically restart itself after a successful execution,
  - and under some conditions will restart even in case of failure.

# Working Examples

- Estimating PI using Monte Carlo simulations
  - The simulations are long-running
  - How to make the program responsive?

# Questions

- JavaFX concurrency
- JavaFX workers, tasks, and services
- JavaFX concurrency and responsive UI design
- Example application
  - In the Sample Programs repository

# More Questions

- Dependencies and design
- Observer pattern and design
- Properties and bindings
- Concurrency, workers, tasks, and services
- More importantly
  - Present a few important problems in (non-trivial) application design
  - What are your solutions to the problems?

# Assignment

- Practice Assignments
  - Mandatory and bonus