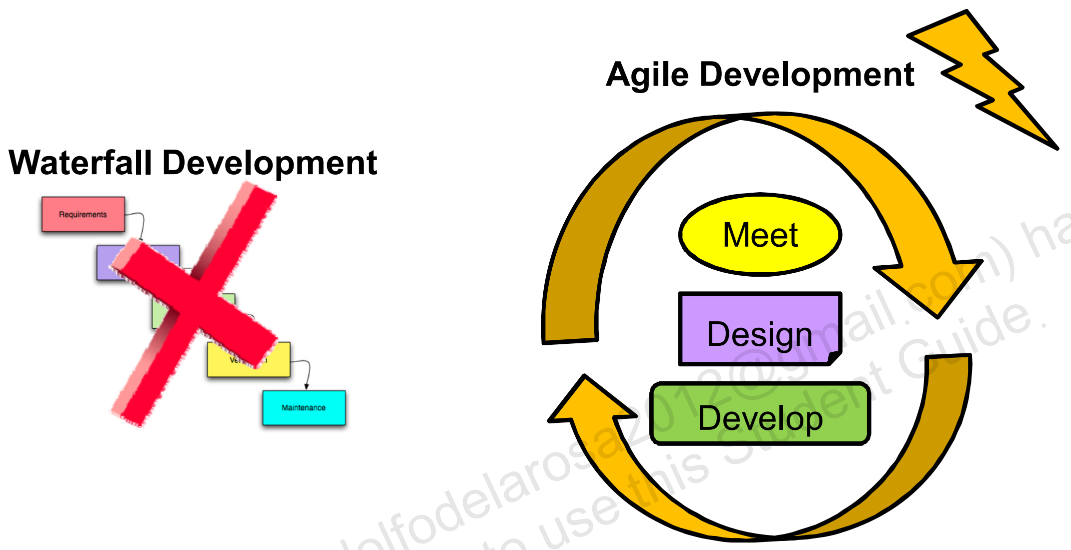
A
Development Methodologies and Design Patterns

Topics

- Agile Development
- Design Patterns

Agile Development

Agile development is a development process that emphasizes frequent status meetings, short-term goals, and good communication.



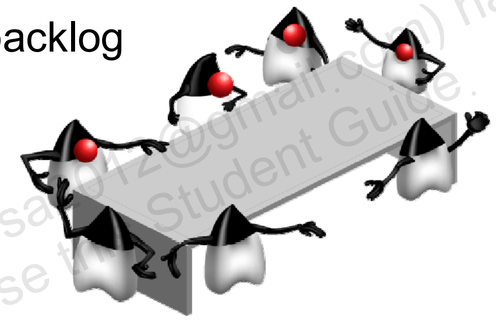
Agile development is a development process that emphasizes frequent status meetings, short-term goals, and good communication. In general, Agile development is a big improvement over the typical Waterfall development. The Waterfall development consists of discrete tasks that must be completed before the new phase can begin. With Agile development, developers do iterative development that lets them complete tasks without focusing on phases. The benefits of Agile development are:

- Reduces project overhead by streamlining process
- Creates better collaboration between team members
- Allows team members to contribute to the best of their abilities
- Promotes iterative development
- Delivers product earlier
- Catches issues sooner
- Receives end-user inputs during development
- Adjusts schedule sooner and creates a better estimate of work

Scrum

Scrum is a type of Agile development methodology, and it has an iterative development style which includes:

- Meeting two to five times a week
- Team members choosing tasks
- Team members working toward short-term goals called sprints
- Integrating builds often
- Putting incomplete tasks in a backlog



A key principle of Scrum is its recognition that during a project, the customers can change their minds about what they want and need, and that unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. The key feature of scrum is that you iterate. So by meeting with customers on a regular basis, and showing them sample code on a regular basis, you avoid any surprises. As such, Scrum adopts an empirical approach—accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the group’s ability to deliver quickly and respond to emerging requirements. Anything in the backlog at the end of the cycle is evaluated for inclusion in future releases.

Advantages:

- Increased speed
- Increased flexibility

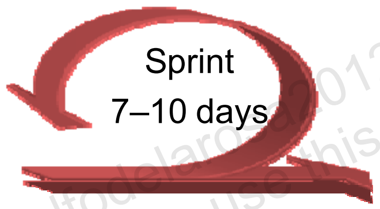
Disadvantages:

- Decreased design
- Decreased quality

Scrum Terminology

Term	Definition
Scrum	• Regularly scheduled meeting to identify and assign tasks, two to five times a week • Traditionally limited to 15–30 minutes
Sprint	• A development iteration • Traditionally a duration of 7–10 days • Designed for iterative tasks
Sprint Goals	• Intermediate goals designed to demonstrate progress toward the high-level milestones • Typically short term goals

Scrum



Sprint Goals

Scrum Roles

Role	Responsibilities
Product Owner	• Helps identify high level goals and milestones • Helps to resolve issues and remove roadblocks if necessary
Scrum Master (Lead Developer)	• Leads scrum meetings • Maintains task list • Adapts to changing requirements and problems • Keeps Project Owner informed
Team Member (Developer)	• Self organized • Volunteers for tasks during each sprint and completes them

Topics

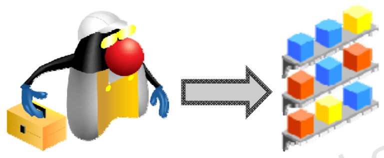
- Agile Development
- Design Patterns

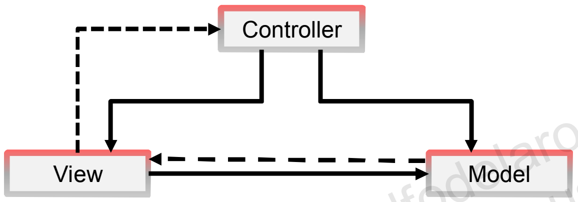
Design Patterns: An Introduction

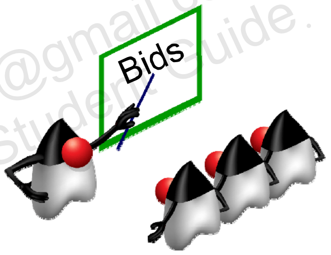
A design pattern is a solution to a common software problem. Object-oriented design patterns show relationships and interactions between objects and classes

Three patterns are used in this course:

- Builder
- Observer
- Model-View-Controller (MVC)







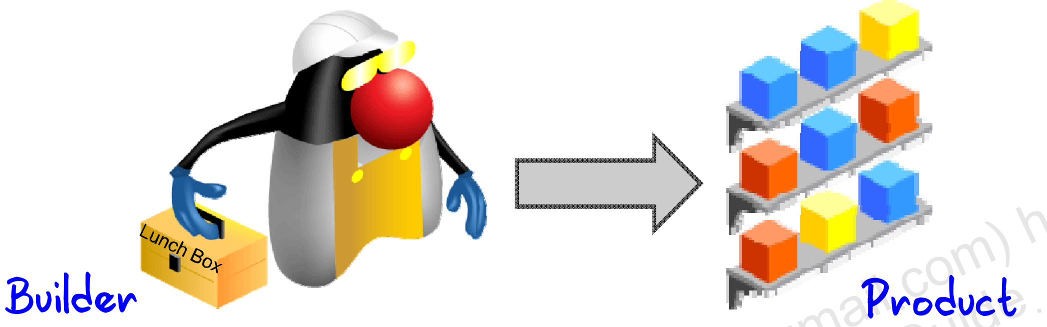
Although described in the early 1960s, design patterns became popular in the 1990s when the Gang of Four introduced design patterns in their book, *Design Patterns: Elements of Reusable Object-Oriented Software*. (The Gang of Four is Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.)

Design patterns provide solutions to common software problems, and they give generalized solutions in the form of templates that can be applied to real-world problems.

- The pattern is a proven and tested solution.
  - Development is more cohesive.
  - Isolates variability.
  - Makes the solution easy to understand.
- The pattern facilitates communication between designers and developers.
  - Both groups will have a mental picture of the high-level design when they think in terms of patterns.
- Can speed up the development process
- Templates can apply to real-world situations.

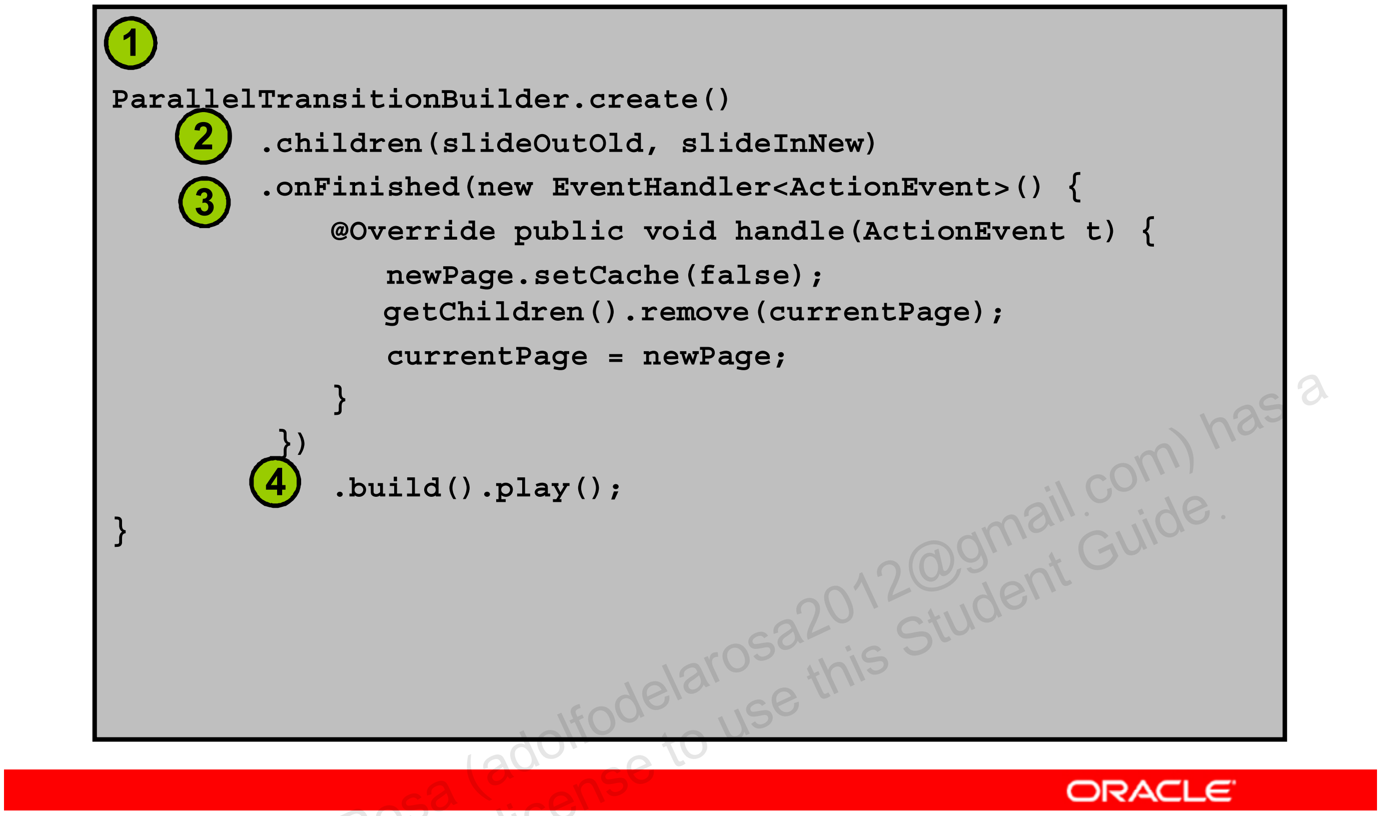
Builder Design Pattern

The Builder pattern separates the construction of a complex object from its representation and simplifies creation of complex objects.



The Builder pattern separates the construction of a complex object from its representation so it provides an easier way to construct complex objects.

Builder Pattern Example



```
1
ParallelTransitionBuilder.create()
    2    .children(slideOutOld, slideInNew)
    3    .onFinished(new EventHandler<ActionEvent>() {
             @Override public void handle(ActionEvent t) {
                newPage.setCache(false);
                getChildren().remove(currentPage);
                currentPage = newPage;
             }
          })
        4  .build().play();
}
```

Implementing the Builder Pattern

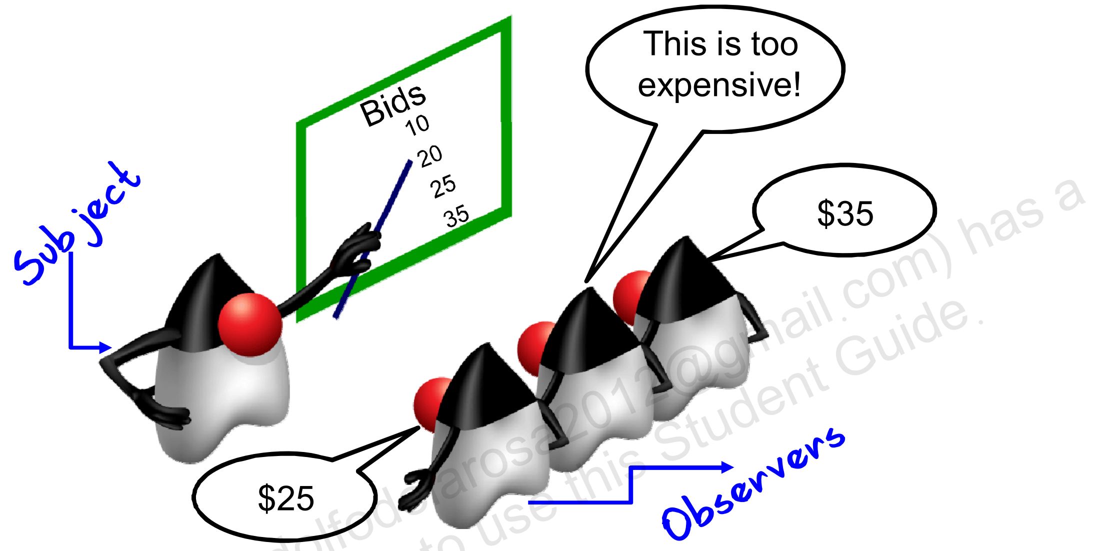
The Builder design pattern is applied to the Henley application animation as the following:

1. `ParallelTransitionBuilder` is the builder class
2. Defines which node is disappearing and what is appearing
3. Creates the event handler that runs when the animation finishes and resets the current page cache
4. Builds the animation and calls the `play()` method

Observer Design Pattern

The Observer pattern has a *subject* that maintains a list of *observers* and notifies them of changes.

Duke's Auction



The Observer pattern is a software design pattern in which an object, called the *subject*, maintains a list of its dependents, called *observers*, and notifies them automatically of any state changes, usually by calling one of their methods. The Observer pattern defines a one-to-many relationship so that when one object changes state, the others are notified and updated automatically. It is mainly used to implement distributed event-handling systems. The Observer pattern is also a key part in the familiar MVC architectural pattern.

Observer Pattern Example

```
final ObservableList<Integer> bidList =     1
               FXCollections.observableArrayList(20, 25, 30);
ListView bidView = new ListView(bidList);     2
final TextField bidField = new TextField();


   bidList.addListener(new ListChangeListener(){   3
      @Override
      public void onChanged(ListChangeListener.Change
change){


sizeField.setText(Integer.toString(bidList.size()));
      }
   }
```

ORACLE

Implementing the Observer Pattern

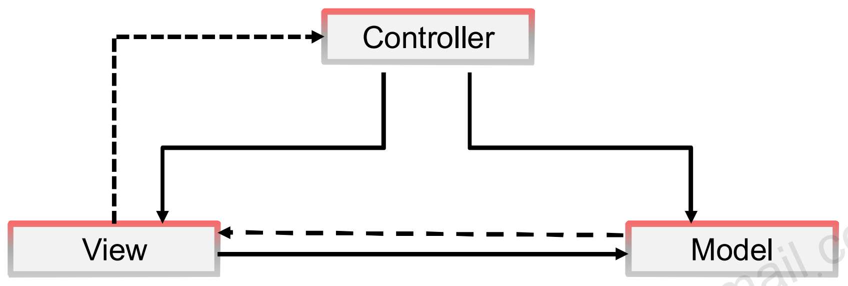
In the code example in the slide, the `bidList` (1) is the subject. The `ListView` (2) and the `ListChangeListener` event handler (3) are observers.

The `ListView` is set up as one listener and automatically updates each time the list is updated.

`ListChangeListener` event handler is a second listener. This displays the size every time you add a new bid. (`size` is the size of the `bidList`.)

The MVC Design Pattern

The pattern isolates the application logic for the user from the user interface.



- The pattern isolates "domain logic" (the application logic for the user) from the user interface (input and presentation), permitting independent development, testing, and maintenance of each (separation of concerns).
- Use of the Model/View/Controller (MVC) pattern results in applications that separate the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.