Design Patterns: Live and In Action!

Alex Mariakakis
CSE 331, Autumn 2013
With material from Krysta Yousoufian, Marty Stepp, Mike Ernst, and others
Example 1: Playing Cards
Card class

```java
public class Card {
    private ___ suit;
    private int rank;
    ...
}
```

- **suit** should be CLUBS, DIAMONDS, HEARTS, or SPADES
- How do we represent this?
int constants

public class Card {
    public static final int CLUBS = 0;
    public static final int DIAMONDS = 1;
    public static final int HEARTS = 2;
    public static final int SPADES = 3;

    private int suit;
    private int rank;
    ...
}

- What’s wrong with this approach?
String constants

```java
public class Card {
    public static final String CLUBS = "CLUBS";
    public static final String DIAMONDS = "DIAMONDS";
    public static final String HEARTS = "HEARTS";
    public static final String SPADES = "SPADES";

    private String suit;
    ...
}
```

- Is this better?
How about a class?

```java
public final class Suit {
    public static final Suit CLUBS = new Suit();
    public static final Suit DIAMONDS = new Suit();
    public static final Suit HEARTS = new Suit();
    public static final Suit SPADES = new Suit();

    private Suit() {}  // no more can be made
}
```

- Is this better?
How about a class?

```java
public final class Suit {
    public static final Suit CLUBS = new Suit();
    public static final Suit DIAMONDS = new Suit();
    public static final Suit HEARTS = new Suit();
    public static final Suit SPADES = new Suit();

    private Suit() {} // no more can be made
}
```

- Is this better?
- Want to list the abstract values without worrying about the representation
The solution: enums

```java
public enum Suit {
    CLUBS,
    DIAMONDS,
    HEARTS,
    SPADES
}
```

- Effective Java Tip #30: “Use enums instead of int constants”
What can you do with an enum?

- Use it as the type of a variable, field, parameter, or return

  ```java
  public class Card {
    private Suit suit;
    ...
  }
  ```

- Compare with `==` (why don’t we need `equals`?)

  ```java
  if (suit == Suit.CLUBS) {
    ...
  }
  ```
What else can you do?

- Get the value’s name (equivalent to `toString`)
  ```java
  // Gets “CLUBS”, “SPADES”, etc.
  suitName = card.getSuit().getName();
  ```

- Compare with `switch` statement

- Lots more, in Java!
  - Enums are actually objects in Java (`ints` in C)
  - Can have fields, methods, and constructors
The switch statement

```c
switch (enum e) {
    case value:
        code;
        break;
    case value:
        code;
        break;
    ...
    default: // if not one of the above values
        code;
        break;
}
```
The switch statement

- Alternative to if/else
- Only works for integral types (e.g. int, char, enum)
- Case can also end with return
- If no break or return, “falls through” into the next case

```c
switch (enum e) {
    case value:
        code;
        break;
    case value:
        code;
        break;
    ...
    default:
        code;
        break;
}
```
Code example

- See package `enum_switch_demo`
Example 2: Book Printer
Example: Book printer

• Hierarchical book class:
  • Book
    • Chapter
      • Paragraph
  • Want an operation to print out the book’s text (title, chapter headings, paragraphs)
  • Where should the print operation go?
Where should the print operation go?

- **Option 1**: In a `DocumentPrinter` class
  - Pros/cons?

- **Option 2**: In `Book` directly
  - Pros/cons?
Where should the print operation go?

- **Option 1: In a `DocumentPrinter` class**
  - Requires `DocumentPrinter` to define the traversal
  - Traversal could be complicated, could change
  - Might need to traverse many types of documents of different structure
  - Duplicates traversal code among printers
Where should the print operation go?

- **Option 2: In Book directly**
  - Limits ability to add new printers (or other operations)

- Is there a third option?
Option #3: Visitor Pattern

- Want to perform some operation on a hierarchical data structure
  - Needs to “visit” every object
  - Operation defined externally
  - But traversal defined internally, not in the operation
How it works

- Visitor’s `visit` method implements the operation
- Data structure’s `accept` method:
  - tells Visitor to `visit` this object
  - calls `accept` on all children
BookPrinter example

- See package `visitor_demo`
Discussion of book visitor

- Pros?
- Cons?
Discussion of book visitor

- Pros?
- Cons?
- Book pretty simple – is it worth isolating the traversal?
  - For this simple example, perhaps not – complicates code
  - But, might use printer with many different types of documents: Textbook, Novel, Magazine, Newspaper, ...
  - Each document would manage its own structure
Discussion of book visitor

- Pros?
- Cons?
- Book pretty simple – is it worth isolating the traversal?
  - For this simple example, perhaps not – complicates code
  - But, might use printer with many different types of documents: Textbook, Novel, Magazine, Newspaper, ...
  - Each document would manage its own structure
- Other visitors besides printers?
  - Word frequency counter
Example 3: News Feed
News Feed

- Real-time news aggregator
- Displays headlines as they arrive
- What classes should we write?
  - How should they communicate?
Push vs. Pull Communication

- **Model** stores and receives information that **View** needs
- How does **View** get this data?
- Pull approach:
- Push approach:
Review: Push vs. Pull

- **Model** stores and receives information that **View** needs.
- How does **View** get this data?
- Pull approach: **View** asks **Model** if it has new data.
- Push approach: **Model** notifies **View** when it has new data.
- How do we choose which to use?
- Which do we want for our news feed?
Observer/Observable

- Design pattern implementing *push* functionality
- Observable **pushes data to** Observers
- Observers **register with** Observable **to get notifications**
In Java

- **Observable** is a class.
- **Observer** is an interface.
- **Observable** pushes out data by calling:
  - `setChanged` (marks that its state has changed)
  - `notifyObservers`
- **Observer** handles new data in **update** method.
Back to News Feed

- See package observer_demo
Discussion of Observer/Observable

- What if Observer needs to post different kinds of events?
- Often used with MVC – use with CampusPaths?
- GUI: ActionListeners