CSCI 136
Data Structures & Advanced Programming

Lecture 4

Fall 2018

Instructors: Bill Lenhart & Bill Jannen

Last Time

- Control structures
  - Branching: if – else, switch, break, continue
  - Looping: while, do – while, for, for – each
- Object oriented programming Basics (OOP)
- Strings and String methods

Today's Outline

- More on Class Types
- Extending Classes & Abstract Classes
- Technique: Randomizing an array
- Miscellaneous Java
  - Static variables and methods
  - Memory management
  - Access control: public, protected, private

Using Strings

- Application: Parsing an XML file of a CD collection
  - XML = eXtensible Markup Language
  - XML is used for many things
  - CD info:

```
<CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <YEAR>1997</YEAR>
</CD>
```

- How can we find and print just the titles?
  - See CDTitles.java
  - java CDTitles < cds.xml

Classes: An Extended Example

- Idea: Implement a class that describes a single playing card (e.g., "Queen of Diamonds")
- Start simple: a single class – BasicCard
- Think about alternative implementations
- Use an *interface* to allow implementation independent coding
- Factor out common features using *abstract classes*
- Use above to create a card deck
- Let's look at BasicCard

Enum Types are Class Types

```
enum Rank { TWO, THREE, FOUR, FIVE, SIX, SEVEN,
       EIGHT, NINE, TEN, JACK, QUEEN, KING, ACE;
}
```

Notes

- Creates an ordered sequence of named constants
- Can find position of an enum value in sequence
  - `int i = r.ordinal(); // r is of type Rank`
- Can get an array of all values in the enum
  - `Rank[] allRanks = Rank.values();`
- Can use in **for** loops
  - `for (Rank r : Rank.values() ) { ... }`
- Can have its own instance variables and methods

Implementing a Card Object

- Think before we code!
- Many ways to implement a card
  - An index from 0 to 51; a rank and a suit, ...
- Start general.
  - Build an *interface* that advertises all public features of a card
  - Not an implementation (define methods, but don't include code)
- Then get specific.
  - Build specific implementation of a card using our general card interface

Start General: Card: An Interface

- What data do we have to represent?
  - Properties of cards
  - How can we represent these properties?
    - There are often multiple options—name some!
- What methods do we need?
  - Capabilities of cards
  - Do we need *accessor* and/or *mutator* methods?

*

A Card Interface

```
public interface Card {

    // Methods - must be public
    public Suit getSuit();
    public Rank getRank();
}
```

Notes

- It seems sketchy to allow a card to change its value
  - Only make accessor methods
- We could make a tediously long enum for all 52 cards, but we won't

Get Specific: Card Implementations

- Now suppose we want to build a specific card object
- We want to use the properties/capabilities defined in our interface
  - That is, we want to *implement* the interface

```
public class CardRankSuit implements Card {
    . . .
}
```

The Enums for Cards

```
public enum Suit {
    CLUBS, DIAMONDS, HEARTS, SPADES; // the values

    public String toString() {
        switch (this) {
        case CLUBS : return "clubs";
        case DIAMONDS : return "diamonds";
        case HEARTS : return "hearts";
        case SPADES : return "spades";
        }
        return "Bad suit!";
    }
}
```

A similar declaration is defined for Rank

A First Card Implementation

```
public class CardSuitRank implements Card {
// instance variables
      protected Suit suit;
      protected Rank rank;
// Constructors
      public CardSuitRank( Rank r, Suit s)
            {suit = r; rank = s;}
// returns suit of card
      public Suit getSuit() { return suit;}
// returns rank of card
      public Rank getRank() { return rank;}
// create String representation of card
      public String toString() {
           return getRank() + " of " + getSuit();}
}
```

A Second Card Implementation

```
public class Card52 implements Card {
// instance variables
      protected int code; // 0 <= code < 52;
// suit is code/13 and rank is code%13
// Constructors
      public Card52( int index ) {code = index;}
// returns suit of card
      public Suit getSuit() {/* see sample code */}
// returns rank of card
      public Rank getRank() {/* see sample code */}
// create String representation of card
      public String toString() {
            return getRank() + " of " + getSuit();
      }
}
```

Improvements to Card52

Add back a constructor with Rank/Suit parameters

```
public class Card52v2 implements Card {
...
  public Card52v2(Rank theRank, Suit theSuit) {
    code = theSuit.ordinal() * 13 + theRank.ordinal();
  }
```

Replace switch statements in "get" methods...

```
  public Suit getSuit() {
        return Suit.values()[ code / 13 ];}
  public Rank getRank() {
        return Rank.values()[ code % 13 ];}
```

...by using values() method to get array of enum values

Demo: PokerDeck.java

Interfaces: Worth Noting

- Interface methods **are always** public
  - Java does not allow non-public methods in interfaces
- Interface instance variables are always **static final**
  - static variables are shared across instances
  - final variables are constants: they can't change value
- Most classes contain constructors; interfaces do not!
- Can *declare* interface objects (just like class objects) but cannot instantiate ("new") them