CS250 Intro to CS II

Spring 2017

Topics

- Virtual Functions
- Pure Virtual Functions
- Abstract Classes
- Concrete Classes
- Binding Time, Static Binding, Dynamic Binding
- Overriding vs Redefining

- Reading: pp. 929-952
- Problems: pp. 925-928 15.9-15.15 (all very good)

Abstract Class

- Consider a base class called Shape that contains a draw function
- Circle, Square, and Line are classes that are derived from Shape, and each one has a unique draw function
- If some kind of array of Shape pointers is maintained, a simple draw command can be sent to each array object invoking the specific draw method for each object type
- We are revisiting this idea

Abstract Class

- An abstract class is a class where the programmer never intends to instantiate an object of the abstract class type
- These classes are typically base classes and are used in an inheritance hierarchy to build more generic derived classes
- Parts of the abstract class are not implemented in the base class; therefore, this logic **MUST** be implemented in the derived class

Pure Virtual Functions

- A class is made abstract by having one or more pure virtual functions associated with the class as follows:
  - `virtual void functionName () = 0;`
- Each derived class must provide its own draw function that overrides the draw function of the abstract class

Abstract Class Example

```
class Shape
{
  public:
    Shape (int x = 0, int y = 0);
    void setX (int);
    void setY (int);
    int getX () const;
    int getY () const;
    virtual void draw () = 0;
    virtual double area () = 0;
  private:
    int mX;
    int mY;
};
```

Concrete Class

- A concrete class is any class that can be instantiated
  - An object of a concrete class can be created

Of Shape, Circle, Square, and Line, which are abstract and which are concrete? Why?

Concrete Class Example

```
class Circle : public Shape
{
  public:
    Circle (int x = 0, int y = 0, double radius = 0);
    void setRadius (double);
    double getRadius () const;
    virtual void draw ();
    virtual double area ();
  private:
    double mRadius;
};
```

Virtual Functions

- A virtual function
  - Allows the derived class the ability to override the function and
  - Must have an implementation
- A pure virtual function
  - Requires the derived class to override the function
  - Cannot have an implementation

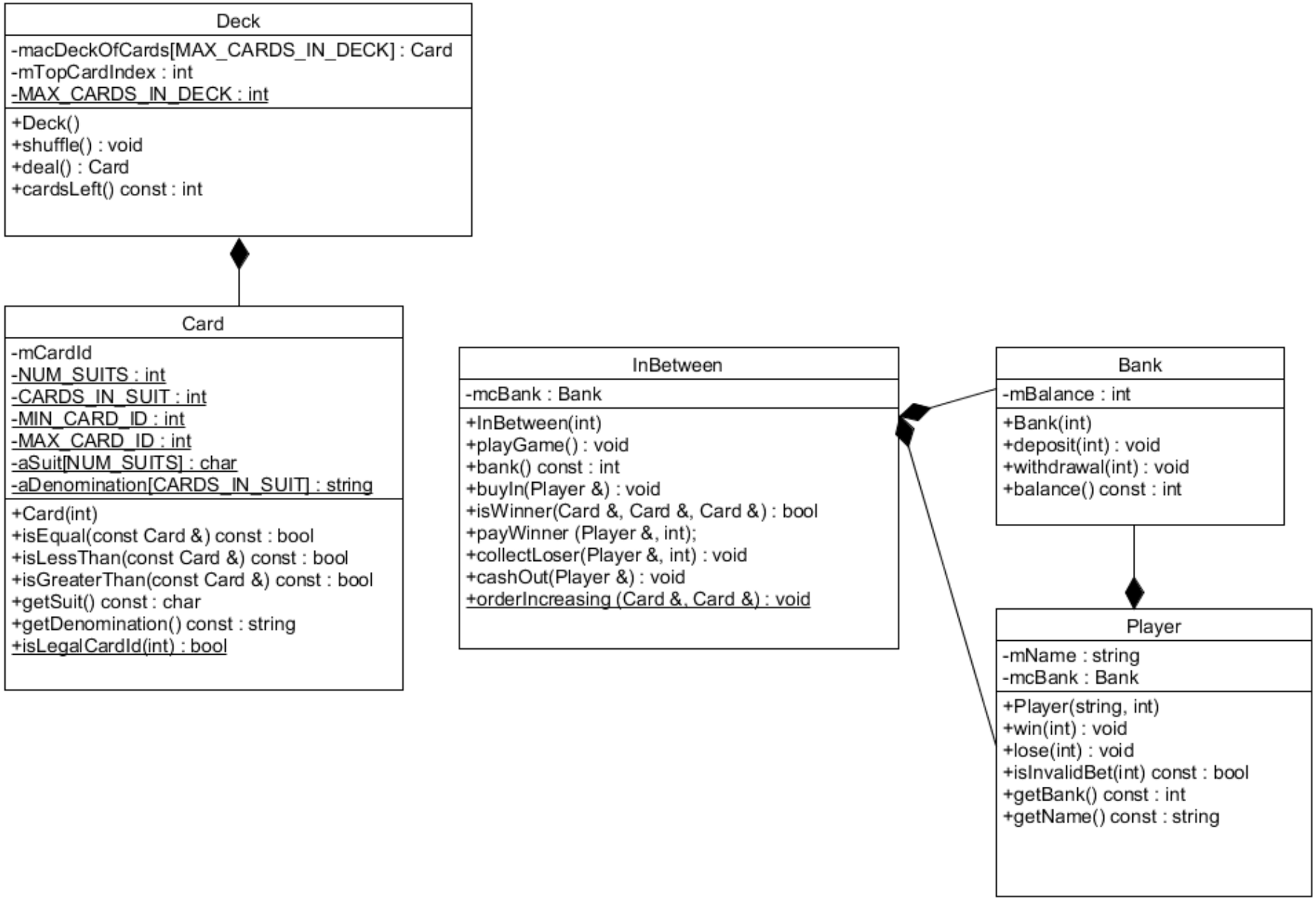
Binding Time

- Binding time - the time at which something becomes known
- Static Binding - binding time that happens during compilation (e.g. a variable's type)
- Dynamic Binding - binding time that happens during runtime (e.g. the heap address of a dynamically allocated memory)

Redefining vs Overriding

- A derived class can “redefine” a base class member (static binding)

- A derived class that redefines a virtual function of a base class is said to “override” the base class function (dynamic binding)


Deck
-macDeckOfCards[MAX_CARDS_IN_DECK] : Card
-mTopCardIndex : int
-MAX_CARDS_IN_DECK : int
+Deck()
+shuffle() : void
+deal() : Card
+cardsLeft() const : int
Card
-mCardId
-NUM_SUITS : int
-CARDS_IN_SUIT : int
-MIN_CARD_ID : int
-MAX_CARD_ID : int
-aSuit[NUM_SUITS] : char
-aDenomination[CARDS_IN_SUIT] : string
+Card(int)
+isEqual(const Card &) const : bool
+isLessThan(const Card &) const : bool
+isGreaterThan(const Card &) const : bool
+getSuit() const : char
+getDenomination() const : string
+isLegalCardId(int) : bool
InBetween
-mcBank : Bank
+InBetween(int)
+playGame() : void
+bank() const : int
+buyIn(Player &) : void
+isWinner(Card &, Card &, Card &) : bool
+payWinner (Player &, int);
+collectLoser(Player &, int) : void
+cashOut(Player &) : void
+orderIncreasing (Card &, Card &) : void
Bank
-mBalance : int
+Bank(int)
+deposit(int) : void
+withdrawal(int) : void
+balance() const : int
Player
-mName : string
-mcBank : Bank
+Player(string, int)
+win(int) : void
+lose(int) : void
+isInvalidBet(int) const : bool
+getBank() const : int
+getName() const : string

The Problem

- Turn InBetween Composition into InBetween Polymorphism
- Three Players:
  - Human
  - RandomAI
  - ConservativeAI

Human

- This player uses the normal keyboard interaction as in InBetweenComposition

RandomAIPlayer

- This player uses the two cards face up to determine a bet amount as follows:
  - rand () % (highCardValue – lowCardValue + 1) + 1
- This player cashes out if a random number mod 11 is equal to 0

ConservativeAIPlayer

- This player uses the two cards face up to determine a bet amount as follows:
  - if the highCardValue – lowCardValue is less than 6, the player bets 1 chip; otherwise, the player bets 2 chips
- This player cashes out if their bank loses or gains 10% of its original value