M1 (b) – Encapsulation

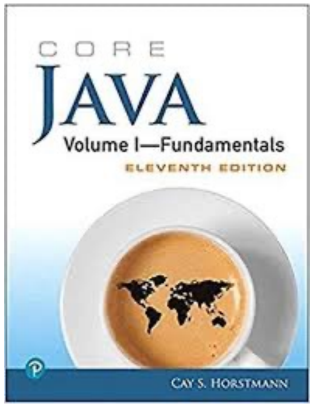
Jin L.C. Guo

Concerns from you (latest survey input)

- Workload
  - Time management
- Format
  - Lectures
  - Lab Tests
- Background

Additional references for Java

- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html



- Core Java Volume I—Fundamentals, Eleventh Edition

- Java: The Complete Reference, Eleventh Edition

Recap of last class

Programming Mechanism Review

- Classes and Interfaces

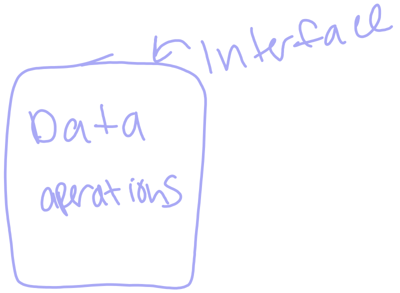
contact point

specification

template for classes who want to implement this interface

blueprint for classes

conceptual / more abstract



Activity 1 :

Code Demo
m1.EscapingReference

only uses immutable Strings
and primatives
so, no

Are there any ways to change the state of an Undergrad object without going through its own methods?

What about Course?

arraylist
refferenced by different
courses, changing
each other

Object Diagram

- Model the structure of the system at a specific time

Object Diagram

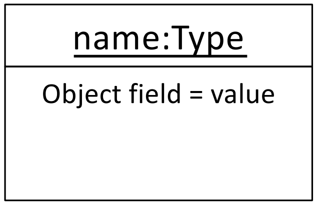
- Model the structure of the system at a specific time

System state

- Complete or part of the system

Object Diagram

- Model the structure of the system at a specific time
- Complete or part of the system
- Include objects and data values



name:Type

Object field = value

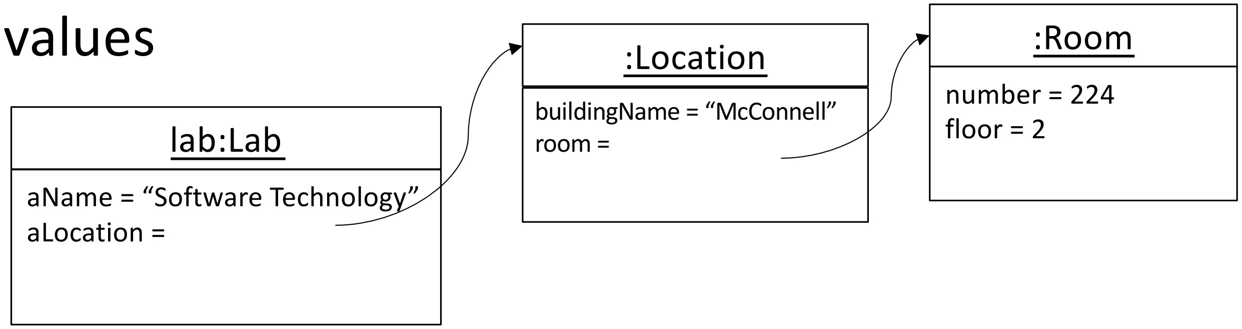
models state
(name and data)

Object Diagram

- Model the structure of the system at a specific time

- Complete or part of the system

- Include objects and data values



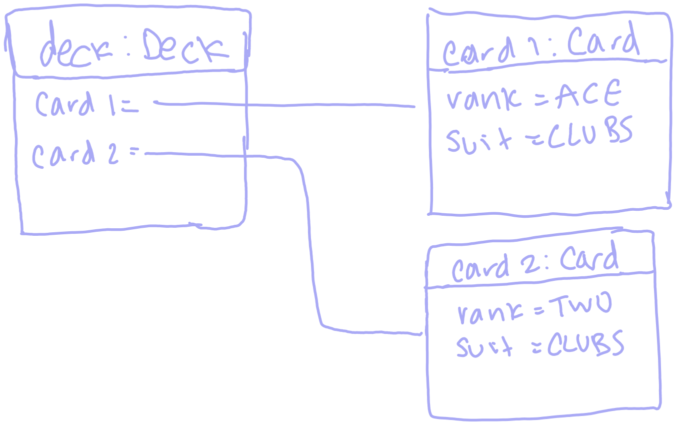
Object Diagram

- Model the structure of the system at a specific time
- Complete or part of the system
- Include objects and data values
- To discover or explain facts of software design (by capturing object relations)

Activity 2 - Draw Object Diagram

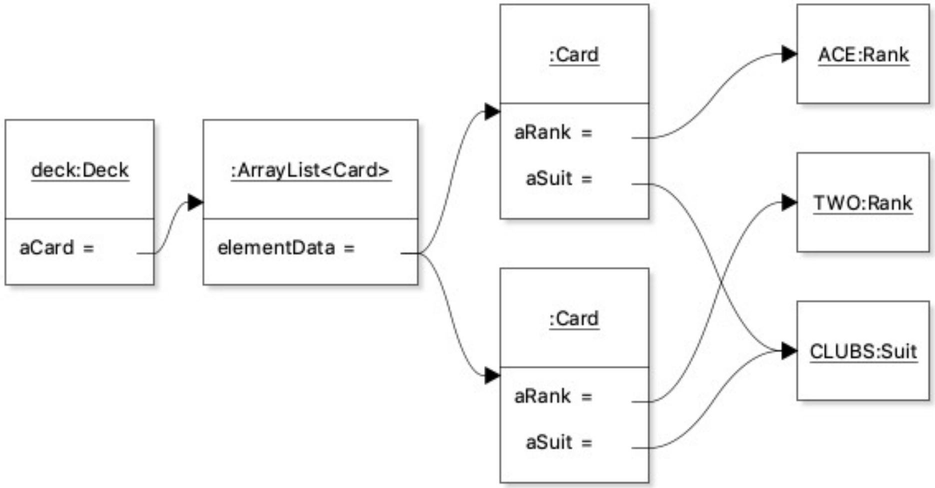
```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        public void addCard(Card pCard)
        {
                aCards.add(pCard);
        }
}
```

```
Deck deck = new Deck();
Card card1 = new Card(Rank.ACE, Suit.CLUBS);
Card card2 = new Card(Rank.TWO, Suit.CLUBS);
deck.addCard(card1);
deck.addCard(card2);
```



Object Diagram - Capturing Object Relations

(enums are reference types)

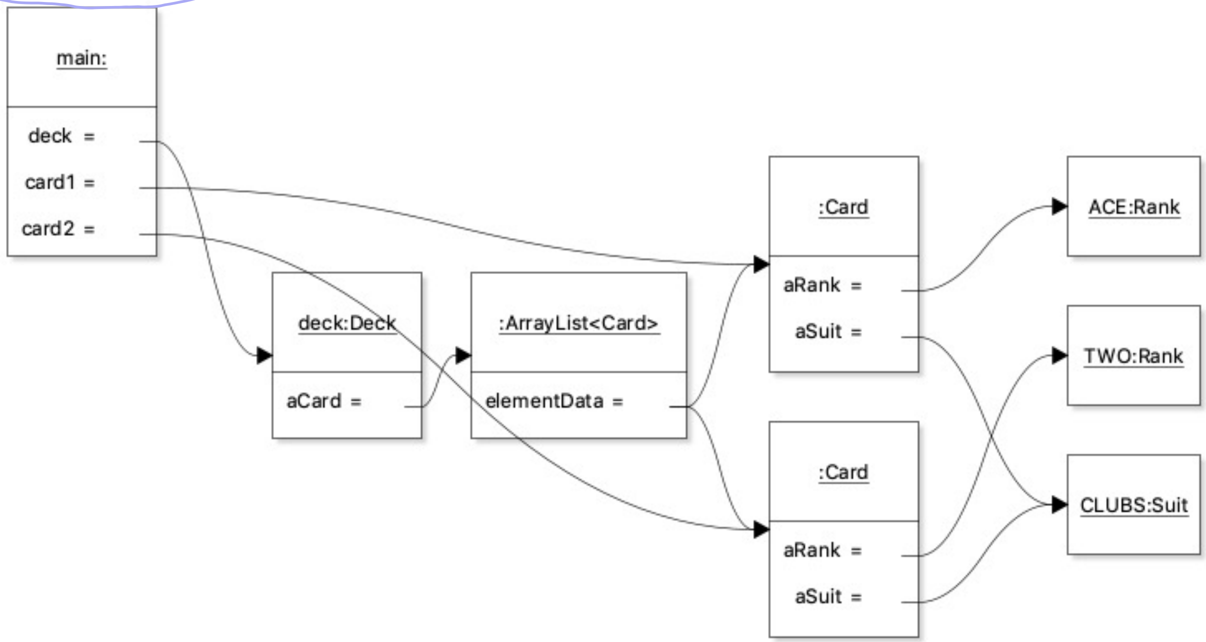


only primitive types written inside box
arrows point to reference types

Capturing Object Relations – Object Diagram

method scope

can show potential escapes



Well-encapsulated Card Class

```
public class Card
{
    final private Rank aRank;
    final private Suit aSuit;

    public Card(Rank pRank, Suit pSuit)
    {
        aRank = pRank;
        aSuit = pSuit;
    }

    public Rank getRank()
    {
        return aRank;
    }

    ……
}
```

What about Deck?

```java
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        public void addCard(Card pCard)
        {
                aCards.add(pCard);
        }


}
```

```java
Deck deck = new Deck();
Card card1 = new Card(Rank.ACE, Suit.CLUBS);
Card card2 = new Card(Rank.TWO, Suit.CLUBS);
deck.addCard(card1);
deck.addCard(card2);
```

What about Deck?

```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …
        public int size ()
        {
                return aCards.size();
        }

        public Card getCard(int pIndex)
        {
                return aCards.get(pIndex);
        }
}
```

```
Card retrievedCard = deck.get(0);
```

card is immutable, so it is safe to share

↑ (because card fields are final)

access one card at a time

Add access methods that only return references to immutable objects.

What about Deck?

```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …

        public List<Card> getCards()
        {
                return new ArrayList<> (aCards);
        }



}
```

```
List<Card> retrievedCards = deck. getCards();
```

Returning a copy

How to make a copy?

- Copy Constructor: a special constructor that creates an object using another object of the same Java class.

```
public Undergrad(Undergrad pUG) {
    this.aID = pUG.aID;
    this.aFirstName = pUG.aFirstName;
    this.aLastName = pUG.aLastName;
}
```

copy each field one by one

How to make a copy?

- Static method within the class

Deck.getCopy(...)

```
public static Undergrad getCopy(Undergrad pUG) {
    Undergrad copy =
       new Undergrad(pUG.aID, pUG.aFirstName, pUG.aFirstName);
    return copy;
}
```

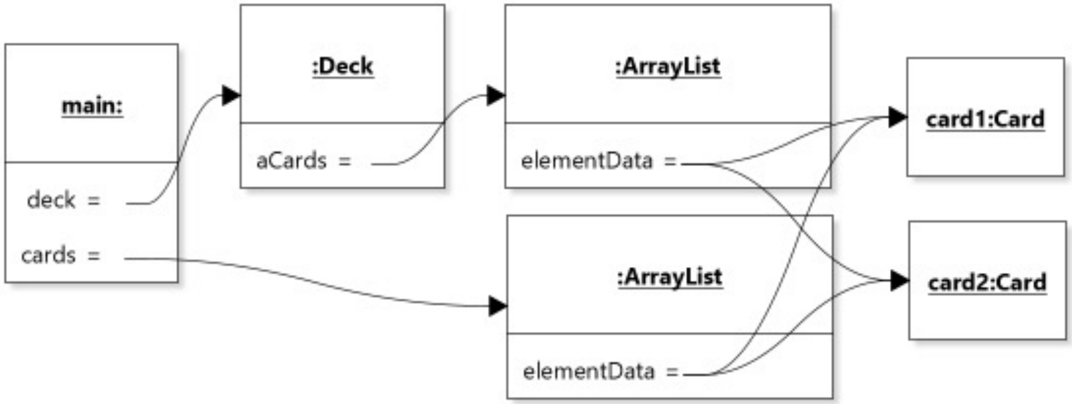
What about Deck?

```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …

        public List<Card> getCards()
        {
                return new ArrayList<> (aCards);
        }


}
```

returns current state

Shallow copy
(cards are shared)

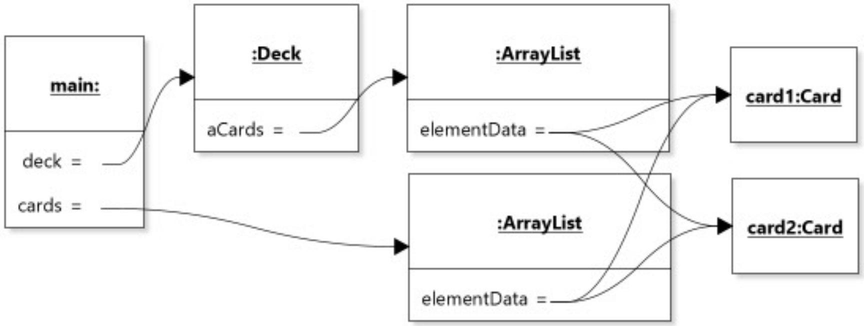
ok because
Cards immutable

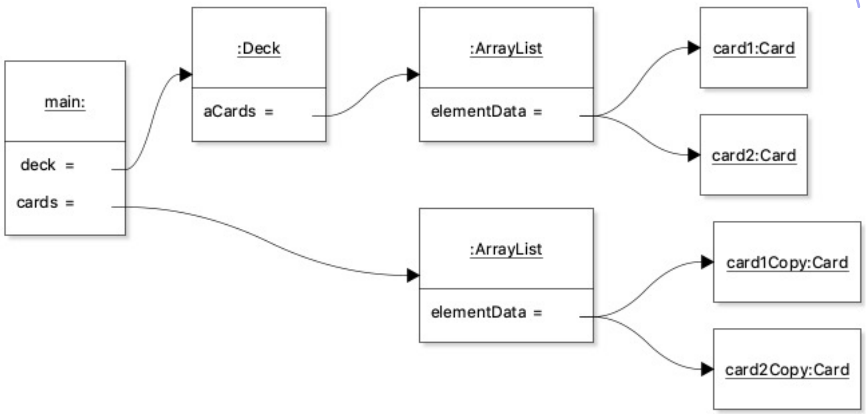


Returning a copy

Shallow Copy VS Deep Copy

creates new objects
for all reference types





if class is mutable - usually
want deep copy

What about Deck?

```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …
        public List<Card> getCards()
        {
            ArrayList<Card> result = new ArrayList<>();
            for(Card card:aCards)
            {
                result.add(new Card(card.getRank(), card.getSuit()));
            }
            return result;
        }
}
```

create an empty list

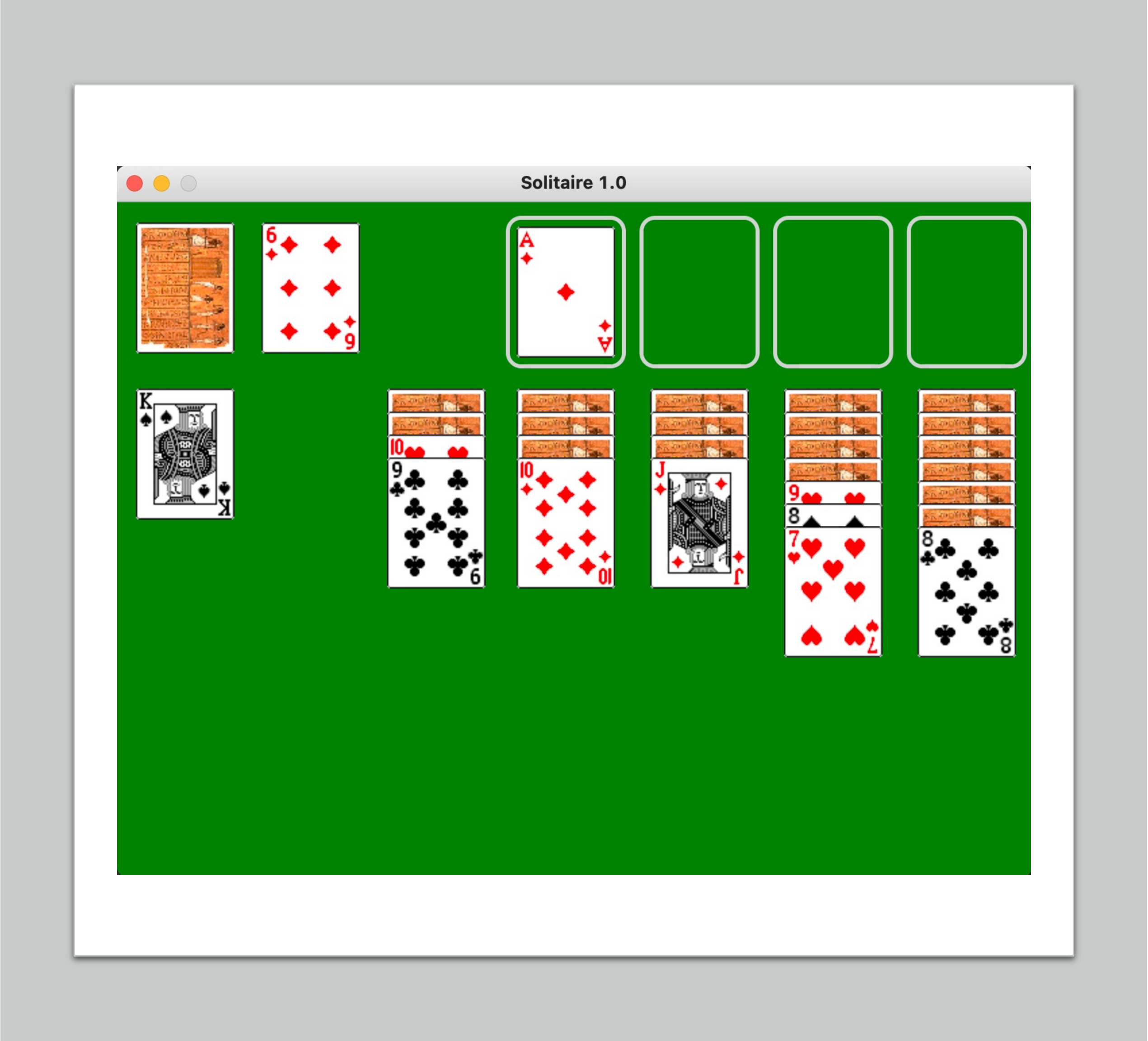
create new card copies and add 1 at a time

Returning a copy

```
public Card(Card pCard){ … … }

public static copyCard(Card pCard){ … … }
```

Activity 3

- Add Color attribute to Card
  - Which class should be changed? suit class
  - What data structure should be used to represent Color? enum



```java
/**
* A card's suit.
*/
public enum Suit
{
    CLUBS, DIAMONDS, SPADES, HEARTS;

    public enum Color {BLACK, RED}

    public Color getColor()
    {
        switch(this)
        {
            case CLUBS:
                return Color.BLACK;
            case DIAMONDS:
                return Color.RED;
            case SPADES:
                return Color.BLACK;
            case HEARTS:
                return Color.RED;
            default:
                throw new AssertionError(this);
        }
    }
}
```

```java
/**
* A card's suit.
*/
public enum Suit
{
    CLUBS(Color.BLACK),
    DIAMONDS(Color.RED),
    SPADES(Color.BLACK),
    HEARTS(Color.RED);

    private Color aColor;

    public enum Color {BLACK, RED}

    Suit(Color pColor)
    {
        this.aColor = pColor;
    }

    public Color getColor()
    {
        return this.aColor;
    }
}
```

package-private/private access

Recap of this module

- Programming mechanisms:
    - Scope and Visibility
- Concepts and Principles:
    - Information Hiding, Encapsulation, Escaping Reference, Immutability
- Design Techniques:
    - Object Diagrams
- Patterns and Antipatterns:
    - Primitive Obsession👎

Next Module

Types and Polymorphism