



M1 (b) – Encapsulation

Jin L.C. Guo

Recap of last class

Activity 1 :

Code Demo
m1.EscapingReference



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



What about Course?

Object Diagram

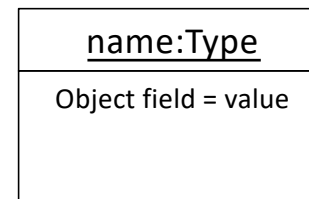
- Model the structure of the system at a specific time

Object Diagram

- Model the structure of the system at a specific time
- Complete or part of the system

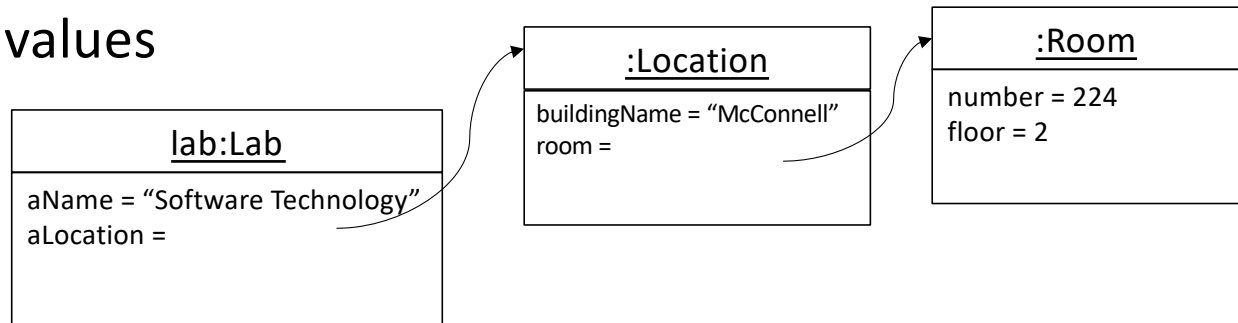
Object Diagram

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



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)

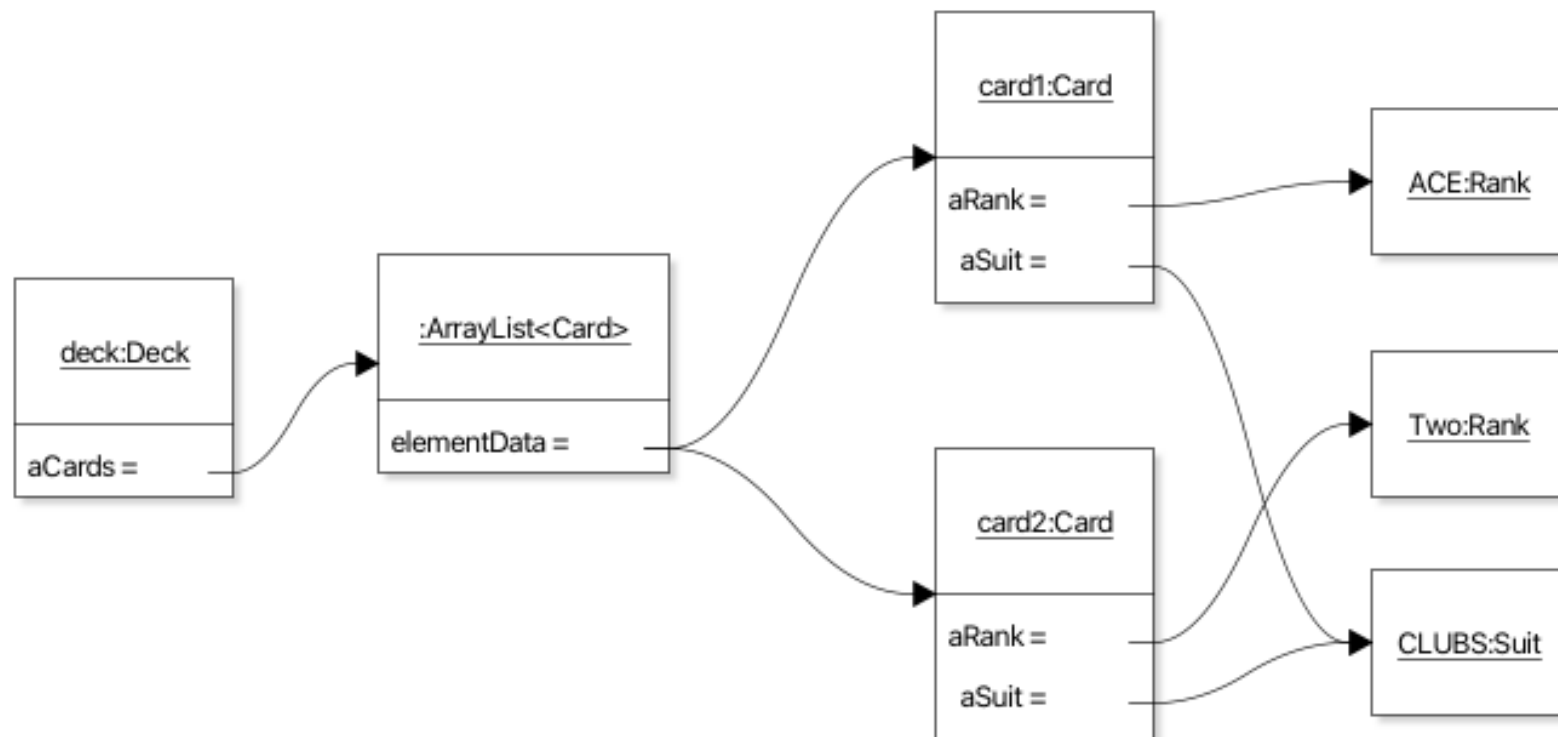
Activity 2 - Draw Object Diagram (Either in paper or in tools such as JetUML)

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

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

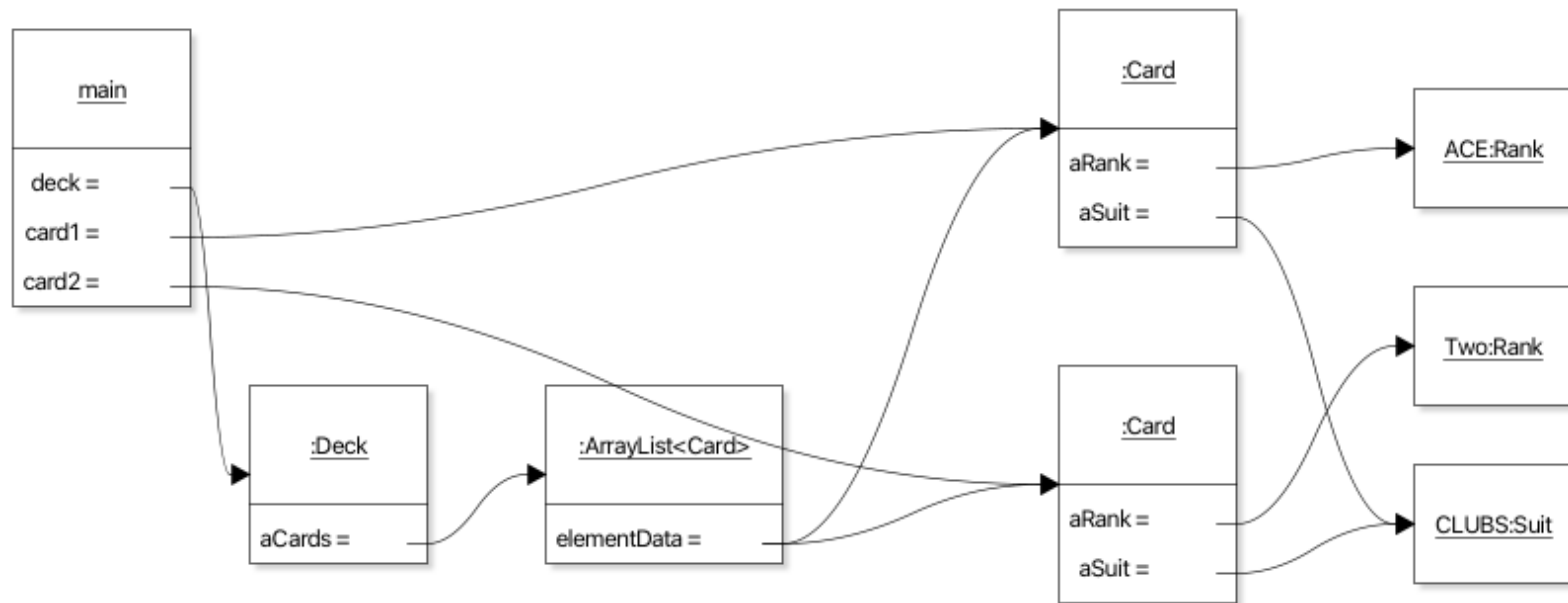
```
Deck deck = new Deck();
Card card1 = new Card(Rank.ACE, Suit.CLUBS);
Card card2 = new Card(Rank.TWO, Suit.CLUBS);
deck.addCard(card1);
deck.addCard(card2);
```

Object Diagram - Capturing Object Relations



Capturing Object Relations – Object Diagram

method scope



Change Card to Immutable

- Immutable: the internal state of the object cannot be changed after initialization.
- How to change the Card Class?

Change Card to Immutable

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

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

    public Rank getRank()
    {
        return aRank;
    }
    public void setRank(Rank pRank)
    {
        aRank = pRank;
    }
    .....
}
```

Change Card to Immutable

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

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

    public Rank getRank()
    {
        return aRank;
    }

    .....
}
```

What about Deck?

```
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);
```

The function requires Deck object to be mutable.

It needs to allow the users to change and query its state.

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);
```

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 Collections.unmodifiableList(aCards);
    }
}
```

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

Returning an unmodifiable view

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) {  
    assert pUG != null;  
    this.aID = pUG.aID;  
    this.aFirstName = pUG.aFirstName;  
    this.aLastName = pUG.aLastName;  
}
```

How to make a copy?

- Static method within the class

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

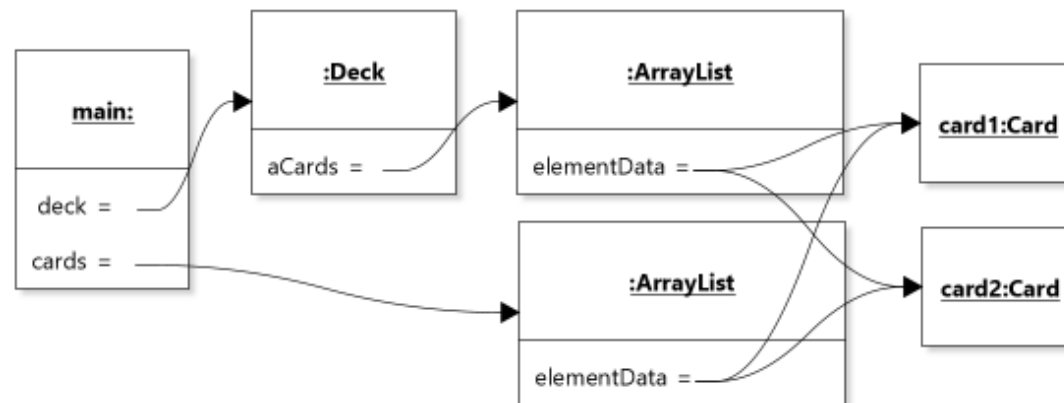
Polymorphic object copying will introduce in M6

What about Deck?

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

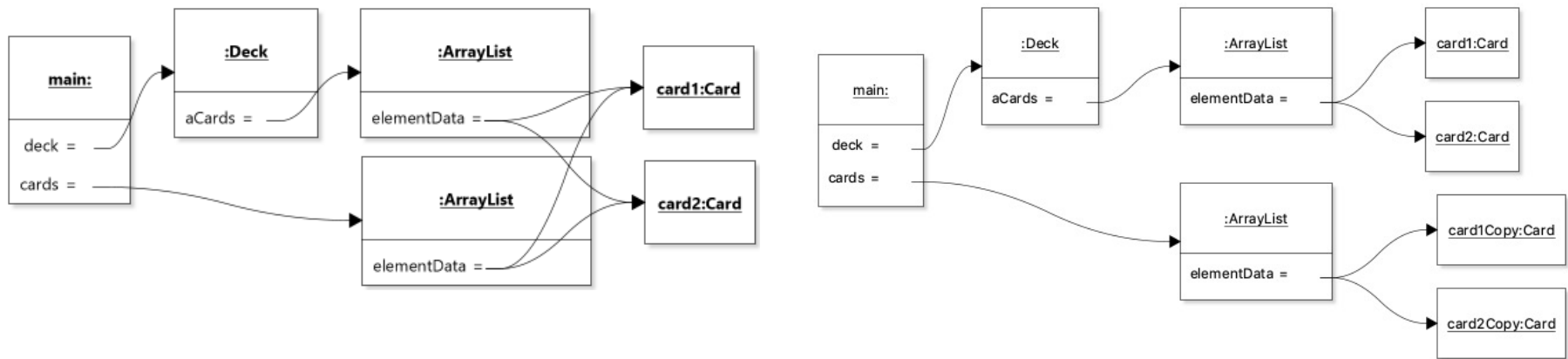
    public List<Card> getCards()
    {
        return new ArrayList<> (aCards);
    }
}
```

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



Returning a copy

Shallow Copy versus 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;
    }
}
```

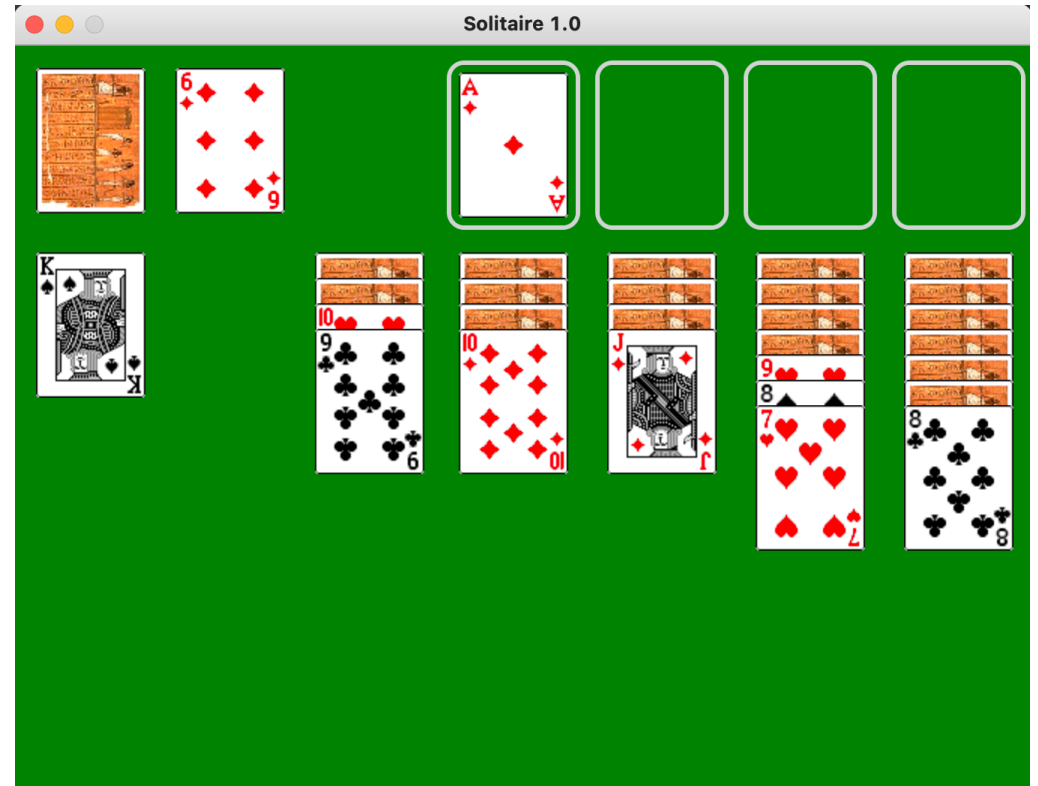
Returning a copy

```
public Card(Card pCard){ ... .. }

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

Activity 3

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




```
/**
 * 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);
        }
    }
}
```

```
/**
 * 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;
    }
}
```

Implicit 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