M1 (b) – Encapsulation

Jin L.C. Guo

Recap of last class

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

Activity 1

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

```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

Activity 2

- Are there any way 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

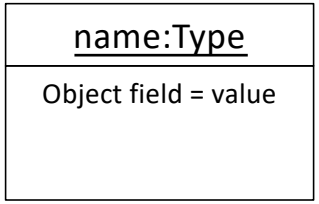
Object Diagram

- Model the structure of the system at a specific time

- Complete or part of the system

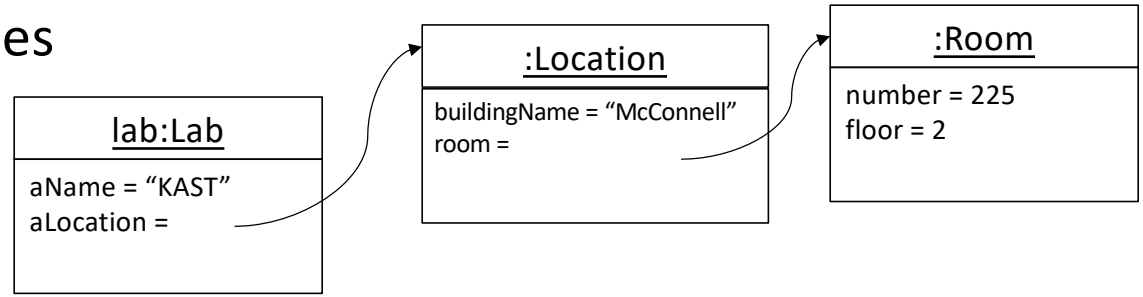
Object Diagram

- 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

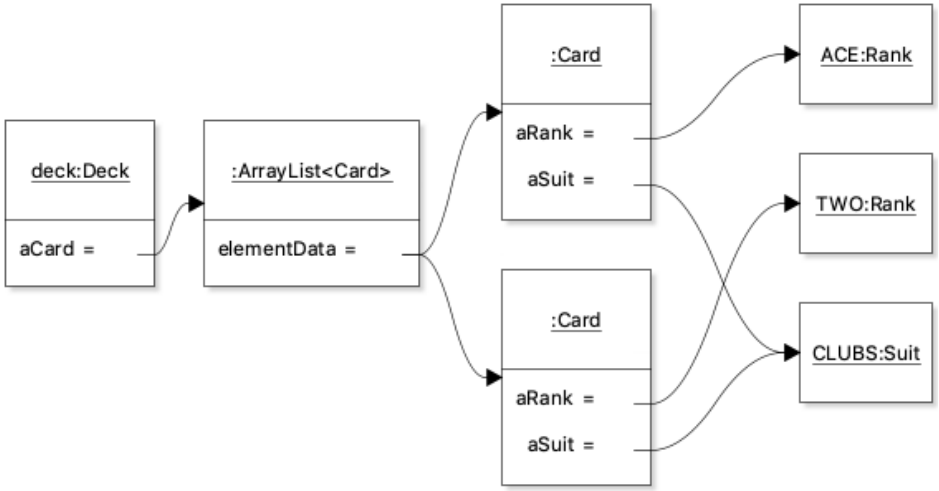
- Model the structure of the system at a specific time
- Complete or part of the system
- Include objects and data values
- To discover or explain facts of software design (by capturing object relations)

Activity 3 - Draw Object Diagram

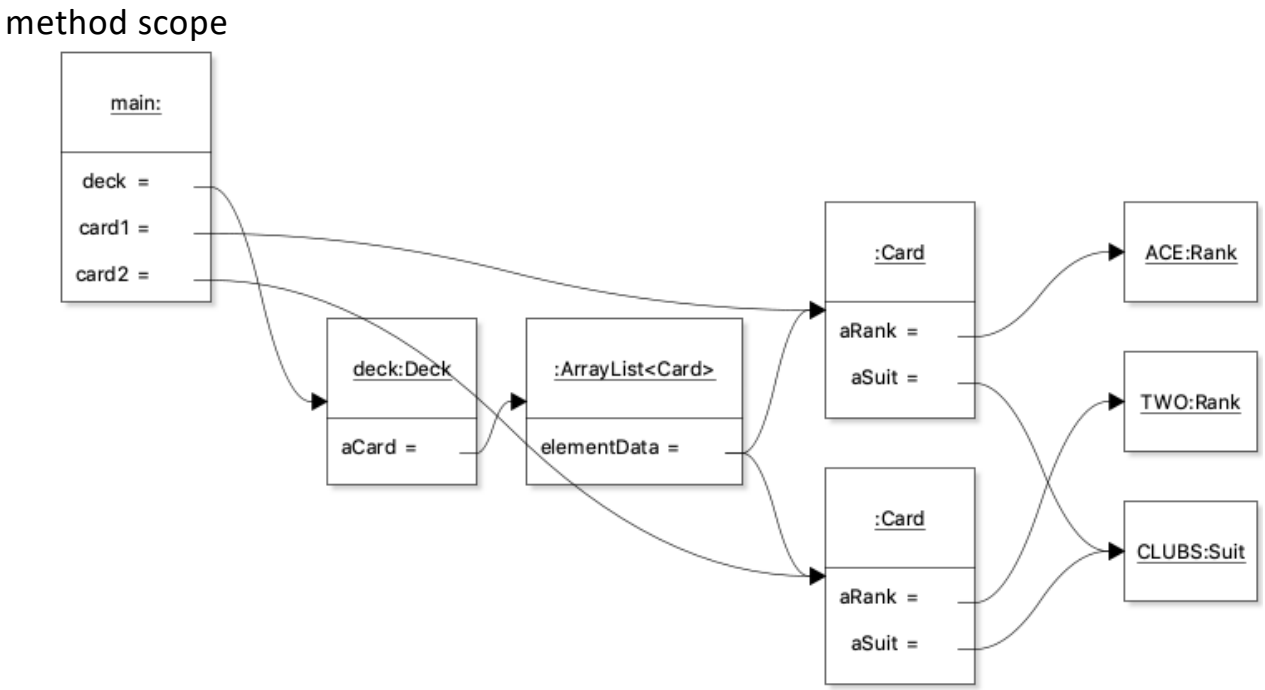
```java
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        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);
```

Object Diagram - Capturing Object Relations



Capturing Object Relations – Object Diagram



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);
        }
}
```

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);
        }



}
```

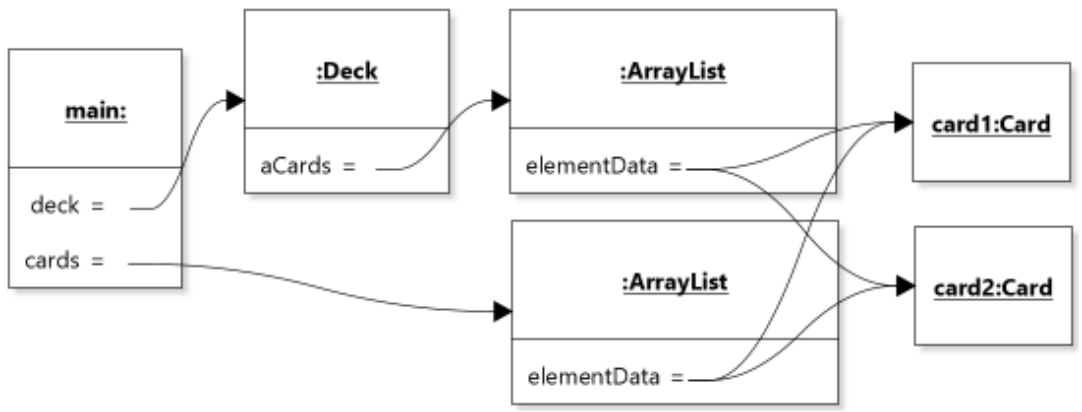
Returning a copy

Copy Constructor

- A special constructor that creates an object using another object of the same Java class.

What about Deck?

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

        … …

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


}
```



Returning a 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){ … … }
```

Shallow Copy VS Deep Copy