M1 (a) – Encapsulation

Jin L.C. Guo

Questions from Last Class

- Lab Test
  - Four sessions
  - Location: Trottier 3120
  - Max 10 people for each time slot

Lab Test Plan

Week	Tuesday Class	Thursday Class	Lab Test
1	Introduction	Encapsulation	
2	Encapsulation	Types and Polymorphism	
3	Types and Polymorphism	Types and Polymorphism	
4	Object State	Object State	Session 1
5	Unit Testing (taught by TAs)	Unit Testing (taught by TAs)	Session 1
6	Composition	Composition	Session 2
7	Inversion of Control	Inversion of Control	Session 2
8	Review	**Midterm (Feb 28th)**	
9	**Study Break**	**Study Break**	
10	Inheritance	Inheritance	Session 3
11	Design Patterns	Design Patterns	Session 3
12	Concurrency	Concurrency	
13	Refactoring	Refactoring	Session 4
14	More Topics in Software Design	More Topics in Software Design	Session 4

Available Slots:

#	Date of the Week	Time
1	Monday	12pm – 1pm
2	Monday	1pm – 2pm
3	Monday	12pm – 1pm
4	Monday	1pm – 2pm
5	Tuesday	12pm – 1pm
6	Tuesday	1pm – 2pm
7	Wednesday	8am – 9am
8	Wednesday	9am – 10am
9	Thursday	8am – 9am
10	Thursday	9am – 10am
11	Friday	2:30pm – 3:30pm
12	Friday	3:30pm – 4:30pm

Questions from Last Class

- Online forum
  - Contributing by: posting, responding, editing, commenting, etc.
- The Goto place when having problems. But check existing posts before posting new ones.

Questions from Last Class

- IDEs and Versions
  - Stick to the required ones (Eclipse, JDK 8, Junit 4) for the exercise and lab test.
  - Try others if you want
    - You can request for creating groups on Piazza.

Questions from Last Class

Maintain the Academic Integrity all the time!

Questions from Last Class

- Midterm conflict with COMP 360

Questions from Last Class

- Missing introduction of myself:

  - PhD in Computer Science and Engineering
  - General Research Interest
    - Software Engineering,
    - Artificial Intelligence
    - Human Computer Interaction
  - Knowledge Enhanced Software Connectivity (KESEC) Lab
    - Software artifact traceability
    - Knowledge Discovery from software artifacts (natural language)
    - Human aspect of Software Engineering
  - http://jguo-web.com

Recap of last class

- The focus and definition of Software Design
- Role of Design in Software Engineering Process
- How to Store and Share Design Knowledge
- Objective of COMP 303

Objectives of this class

- Information Hiding and Encapsulation
- Scope and Visibility
- Object Diagrams
- Escaping Reference
- Immutability

Information Hiding

- *On the criteria to be used in decomposing systems into modules*

David Parnas - Communications of the ACM, 1972 - dl.acm.org

- A principle to divide any piece of equipment, software or hardware, into modules of functionality.

Information Hiding

- *On the criteria to be used in decomposing systems into modules*

David Parnas - Communications of the ACM, 1972 - dl.acm.org

- A principle to divide any piece of equipment, software or hardware, into modules of functionality.

- Modularization can improve the flexibility and comprehensibility of a system while allowing the shortening of its development time.

Very first task (Activity 1)

- Design the representation of a deck of playing cards.

Code under design

Client code

Options: using primitive data types

Use integer

- Clubs 0-12
- Hearts 13-25
- Spades 26-38
- Diamonds 39-51

```
int card = 13; // The Ace of Hearts
int suit = card / 13; // 1 = Hearts
int rank = card % 13; // 0 = Ace
```

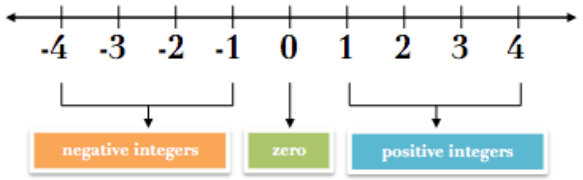
Options: using primitive data types

- Use pair of values [int, int]
  - Rank 0-12
  - suit 0-4

```
int[] card = {1,0}; // The Ace of Hearts
int suit = card[0];
int rank = card[1];
```

Problems?

Representation



Domain Concept

Activity 2

- Representing phone number with string?

- Note: the String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such. [Java Primitive Data Types]

Anti-pattern

- Primitive Obsession
  - **Symptoms**

  Use of primitives for “simple” tasks (such as currency, ranges, special strings for phone numbers, etc.)

Anti-pattern

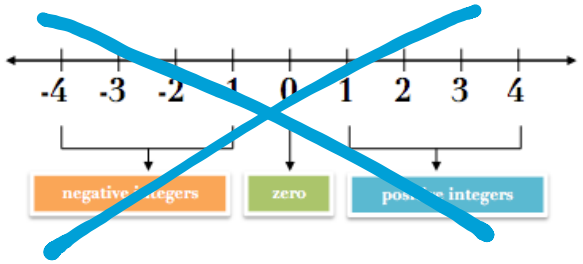
- Primitive Obsession
  - **Symptoms**

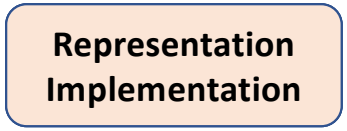
  Use of primitives for “simple” tasks (such as currency, ranges, special strings for phone numbers, etc.)

  - **Treatment**

  Replace Primitive with Object (if you are doing things other than simple printing)

Representation
Implementation





loosely coupled

Define our own Card type

```
public class Card
{

        …

}
```

Characterizing the Card

int constant? string constant?

- Suit
  - Clubs, Hearts, Spades, Diamonds
- Rank
  - Ace, Two, …, Jack, Queen, King

Characterizing the Card

- Suit
  - Clubs, Hearts, Spades, Diamonds

```
public enum Suit
{ CLUBS, DIAMONDS, SPADES, HEARTS
}
```

- Rank
  - Ace, Two, …, Jack, Queen, King

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

Java Enum Type

- For predefined constants
- Instance-controlled -- classes that export one instance for each enumeration constant via a public static final field
- Compile-time type safety

  Suit can only be one of *CLUBS, DIAMONDS, SPADES, HEARTS*
- Add methods and other fields

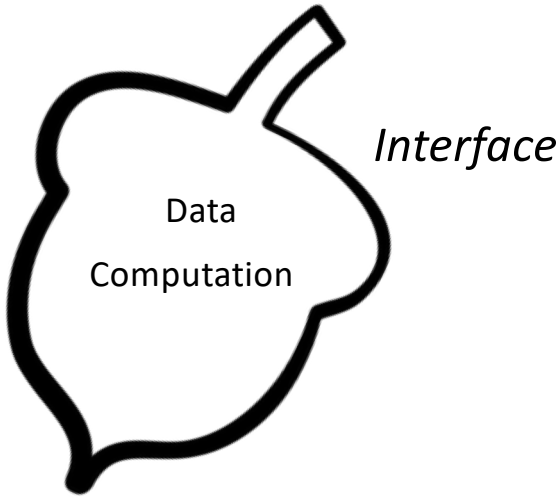
Back to our Card Class

```
public class Card
{
	public Rank aRank;
	public Suit aSuit;
}
```

```
card.aRank = null;
System.out.println(card.aRank.toString());
```

java.lang.NullPointerException

Encapsulation



Goal: to minimize the contact points

Access Modifiers for class members

- Private: accessible from top-level class where it is declared
- Package-private (default): from any class in the package
- Protected: from subclass and any class in the package
- Public: anywhere

Better Encapsulated Card Class

```
public class Card
{
    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;
    }

    ……
}
```

Representation of Deck?

```
List<Card> Deck = new ArrayList<>();
```

Representation of 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);
```

Object Diagram

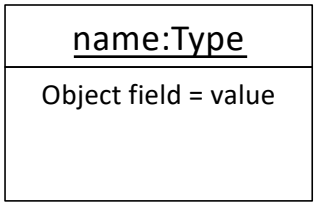
- Model the structure of the system at a specific time

Object Diagram

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

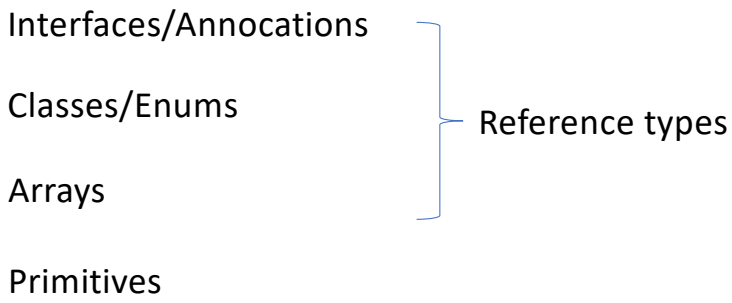
Object Diagram

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



Recall

- Java type system



Recall

- Java Memory Organization

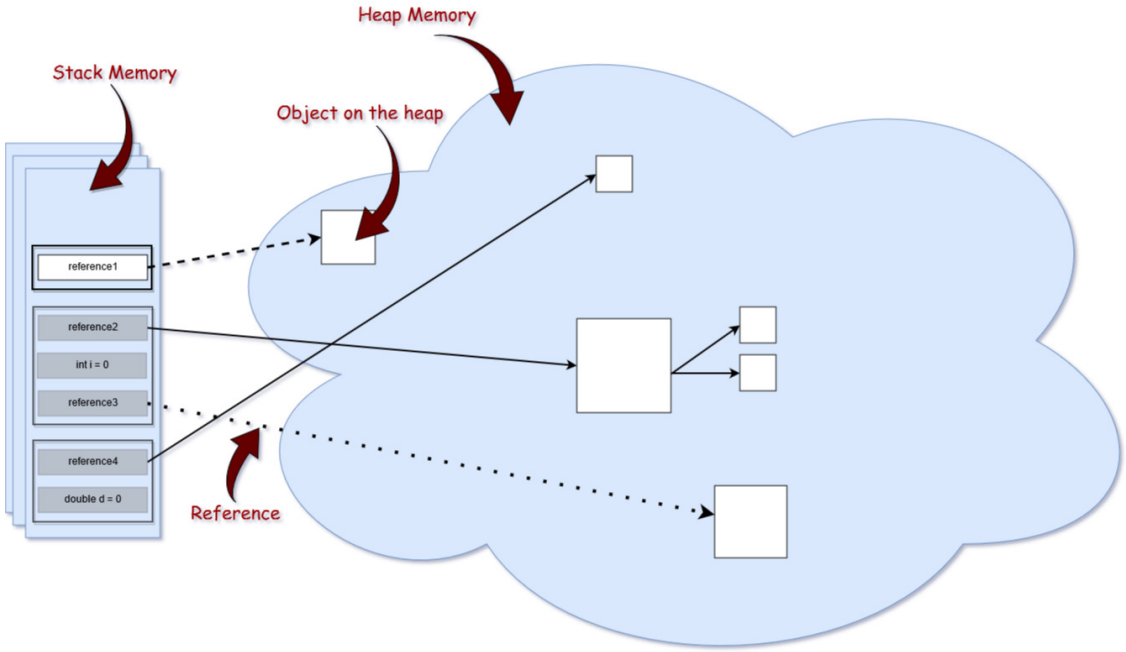
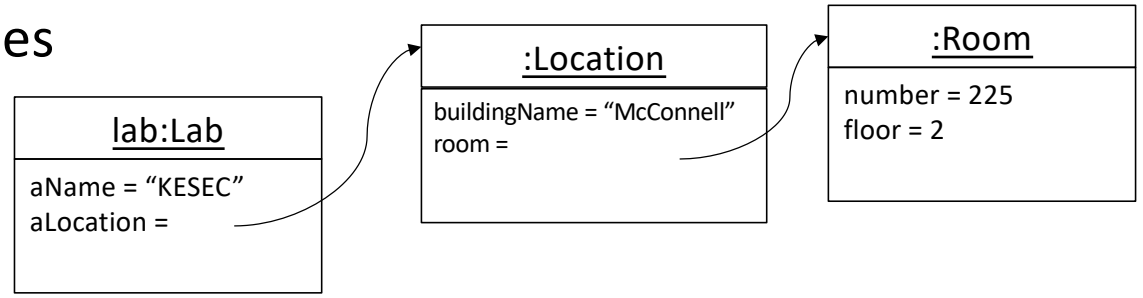


Image Source: https://dzone.com/articles/java-memory-management

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

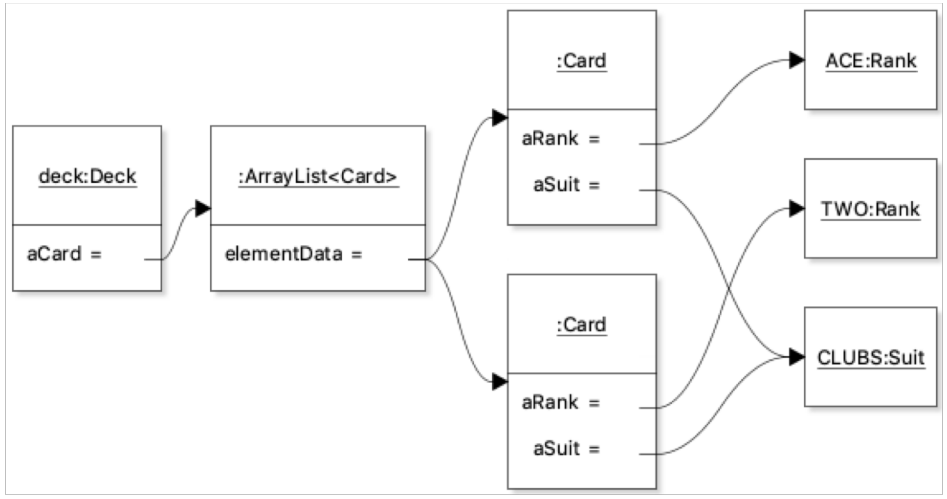
- Model the structure of the system at a specific time
- Complete or part of the system
- Include objects and data values
- To discover or explain facts of software design (by capturing object relations)

Activity 3 - Draw Object Diagram

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

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

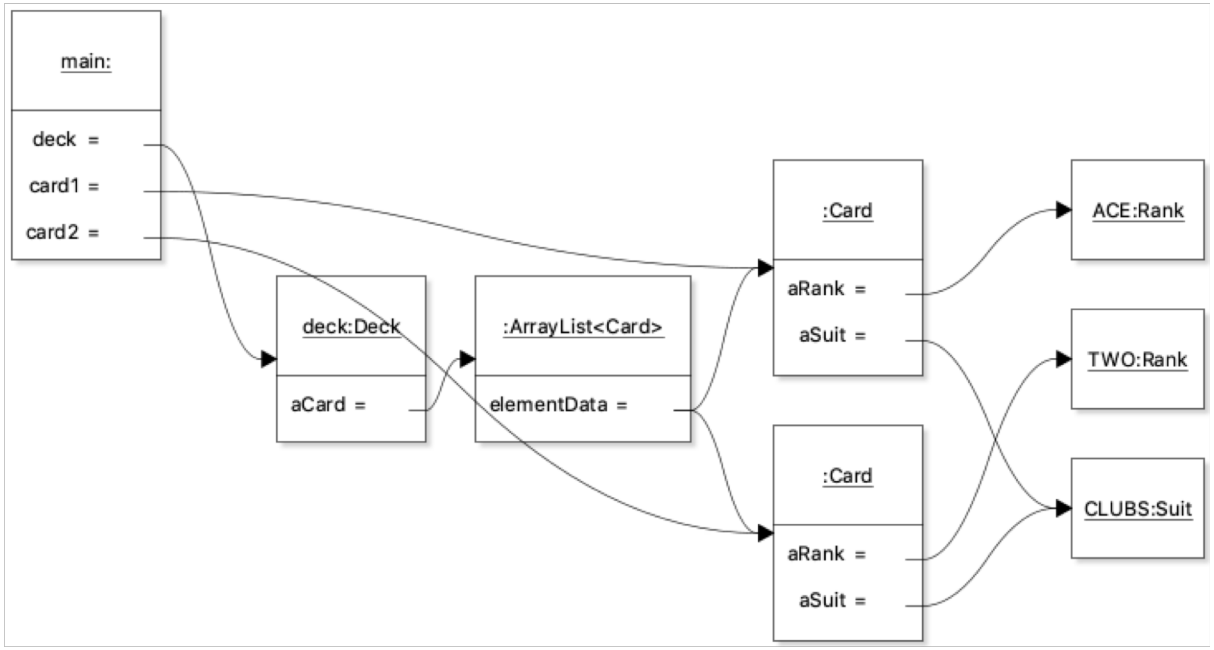
```
Deck deck = new Deck();
Card card1 = new Card(Rank.ACE, Suit.CLUBS);
Card card2 = new Card(Rank.TWO, Suit.CLUBS);
deck.addCard(card1);
deck.addCard(card2);
```

Object Diagram - Capturing Object Relations



Capturing Object Relations – Object Diagram

method scope



Escaping References

- Why this is bad?

```
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);
```

Escaping References

- It should NOT be possible to change the state of an object without going through it's own methods.

- Red flag:

Storing an external reference internally!

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

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

Escaping References

- It should NOT be possible to change the state of an object without going through it's own methods.

- Red flag:

Returning a reference to an internal object!

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

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

Escaping References

- It should NOT be possible to change the state of an object without going through it's own methods.

- Red flag:

Leaking references through Shared structures!

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

        public void collect(List<Card> pAllCard)
        {
                pAllCard.addAll(aCards);
        }
}
```

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
{
    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
{
    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?

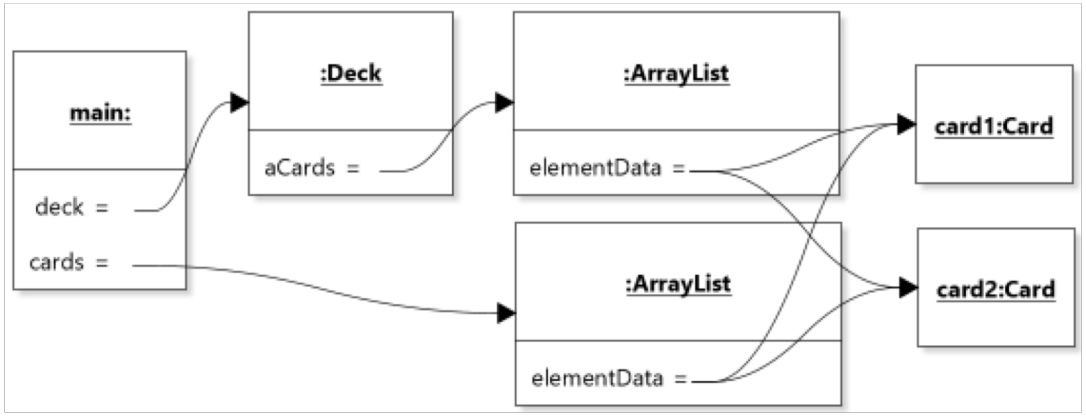
```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …
        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.

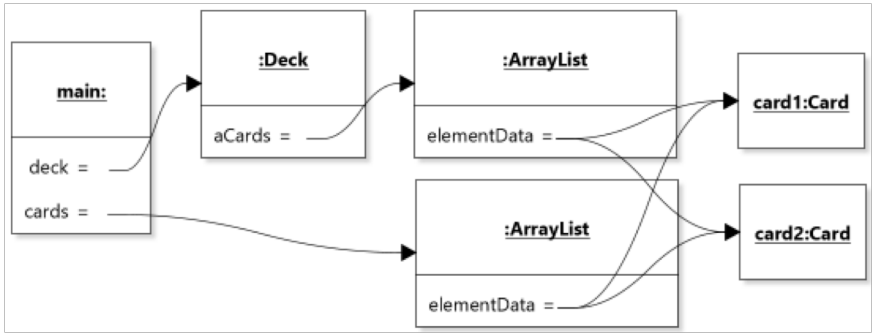
What about Deck?

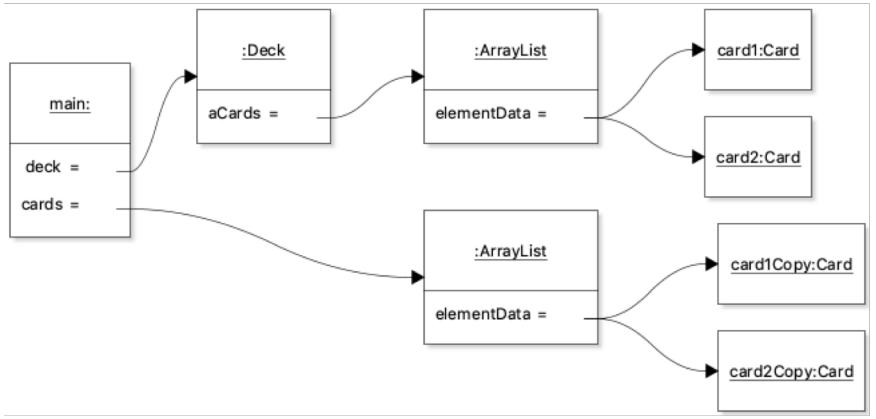
```
public class Deck
{
        private List<Card> aCards = new ArrayList<>();

        … …

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


}
```

Returning a copy



Shallow Copy VS 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 4: Add Color attribute to Card