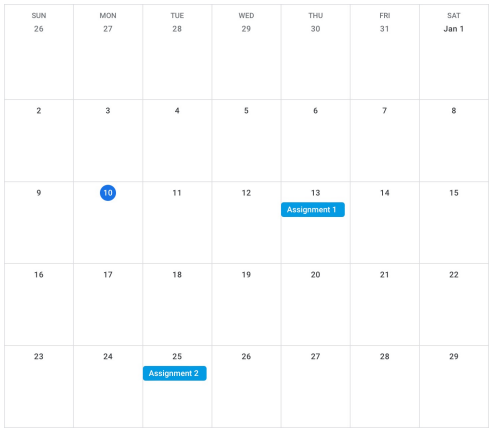
M1 (a) – Encapsulation

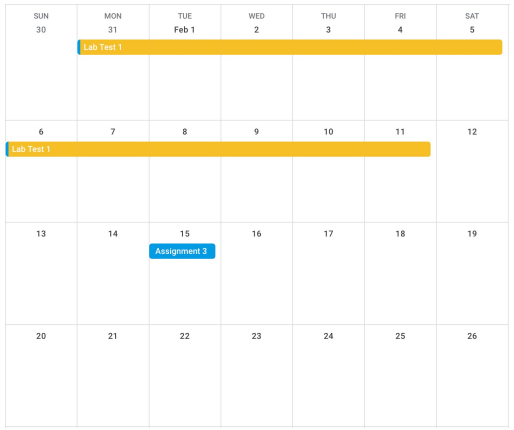
Jin L.C. Guo

Assessment Schedule (Tentative)

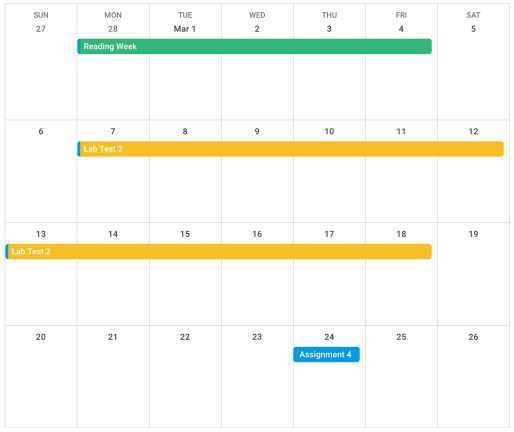
January



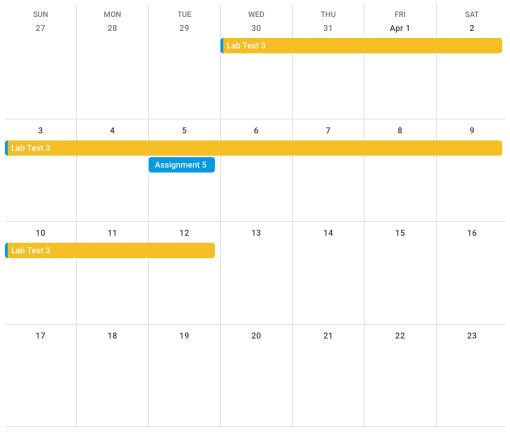
February



March



April



Lab Test

- Three sessions
- Format: In-person (TR3120)
- Max 8 people for each time slot

Available Slots:

We will announce how to sign up the interactive assessment session later.

Slot	Date of the Week	Time
1	Monday	9am - 10am
2	Monday	10am - 11am
3	Monday	1:30pm - 2:30pm
4	Monday	2:30pm - 3:30pm
5	Tuesday	9am - 10am
6	Tuesday	10am - 11am
7	Tuesday	11am - 12pm
8	Tuesday	12pm - 1pm
9	Tuesday	3:30pm – 4:30pm
10	Tuesday	4:30pm – 5:30pm
11	Wednesday	9am - 10am
12	Wednesday	10am - 11am
13	Wednesday	3:30pm – 4:30pm
14	Wednesday	4:30pm – 5:30pm
15	Thursday	9am - 10am
16	Thursday	10am - 11am
17	Thursday	3:30pm – 4:30pm
18	Thursday	4:30pm – 5:30pm
19	Friday	9am - 10am
20	Friday	10am - 11am
21	Friday	11am - 12pm
22	Friday	12pm - 1pm

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 Module

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

Very first task (Activity 1)

- Design the representation of a deck of playing cards.

Code under design

Client code

Programming Mechanism Review

- Java static type system

Interfaces/Annotations
Classes/Enums
Arrays
} Reference types

Primitives *byte, short, int, long, float, double, boolean, char*

Java Memory Organization

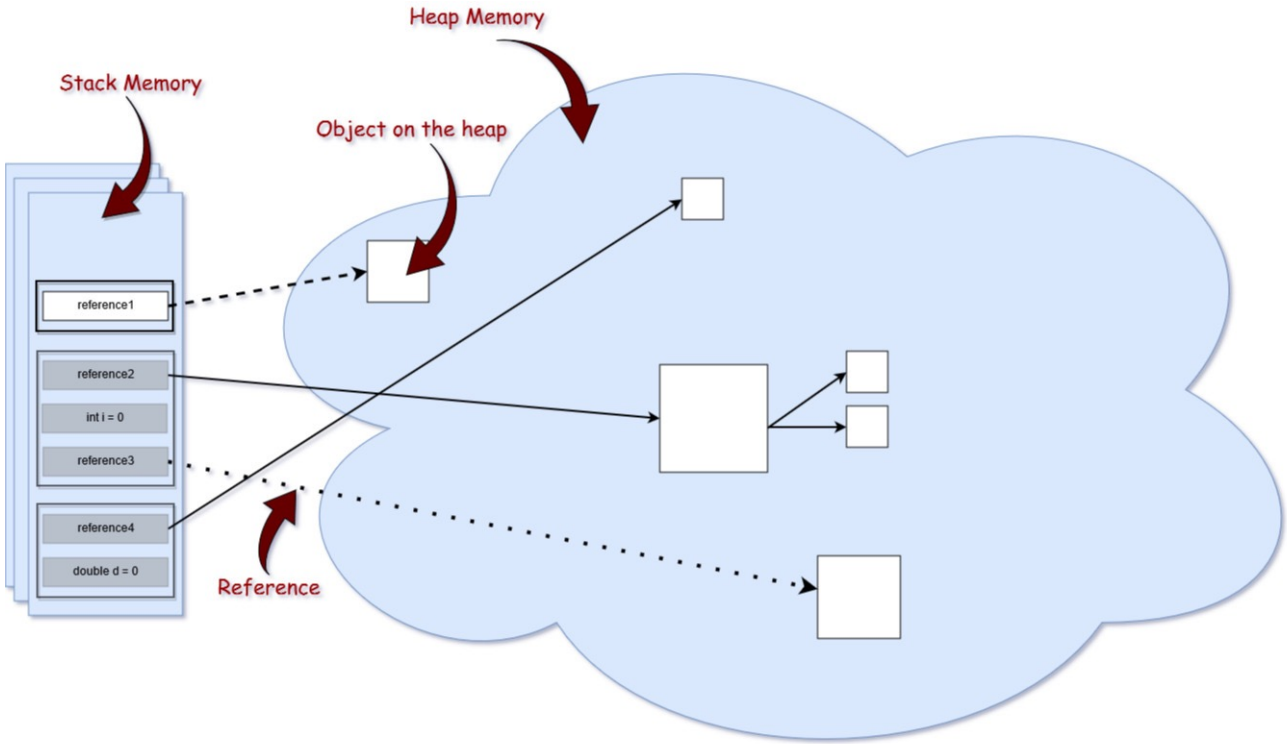


Image Source: https://dzone.com/articles/java-memory-management

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
```

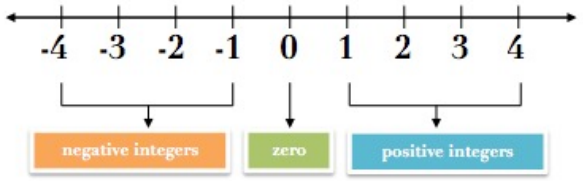
Options: using arrays

- 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

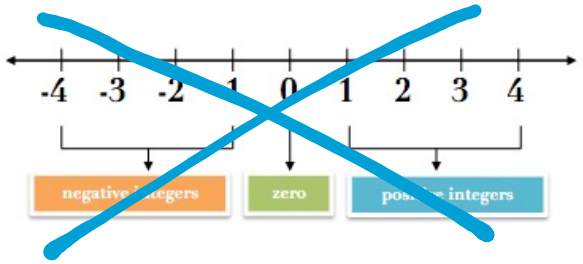
- Primitive Obsession
  - **Symptoms**

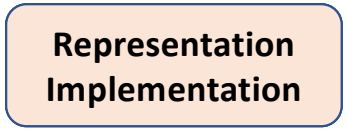
    Use of primitives for “simple” tasks (such as currency, ranges, special strings for phone numbers, etc.)

  - **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
- Compile-time type and value safety

Current Focus

Suit can only be one of *CLUBS, DIAMONDS, SPADES, HEARTS*

- Add methods and other fields

- Instance-controlled -- classes that export one instance for each enumeration constant via a public static final field

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

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

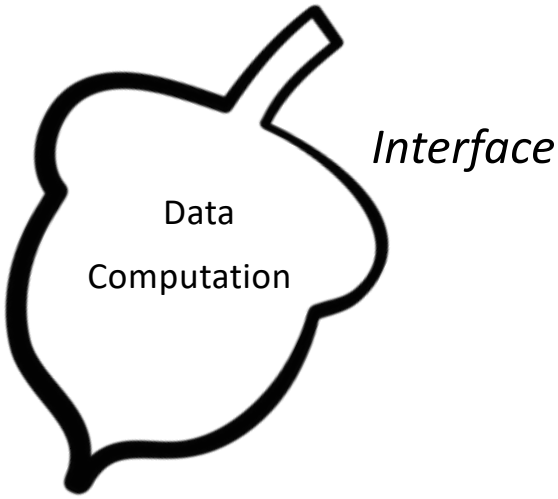
```
public class Card
{
    private Rank aRank;
    private Suit aSuit;

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

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

    ……
}
```

Encapsulation



Goal: to minimize the contact points

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);
```

Information Hiding

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

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

We have tried to demonstrate by these examples that it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing. To achieve an efficient implementation we must abandon the assumption that a module is one or more subroutines, and instead allow subroutines and programs to be assembled collections of code from various modules.

Information Hiding

- 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.

Information Leaking:

- 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 its 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 its 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 its 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 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);
```

Programming Mechanism Review

- Classes and Interfaces