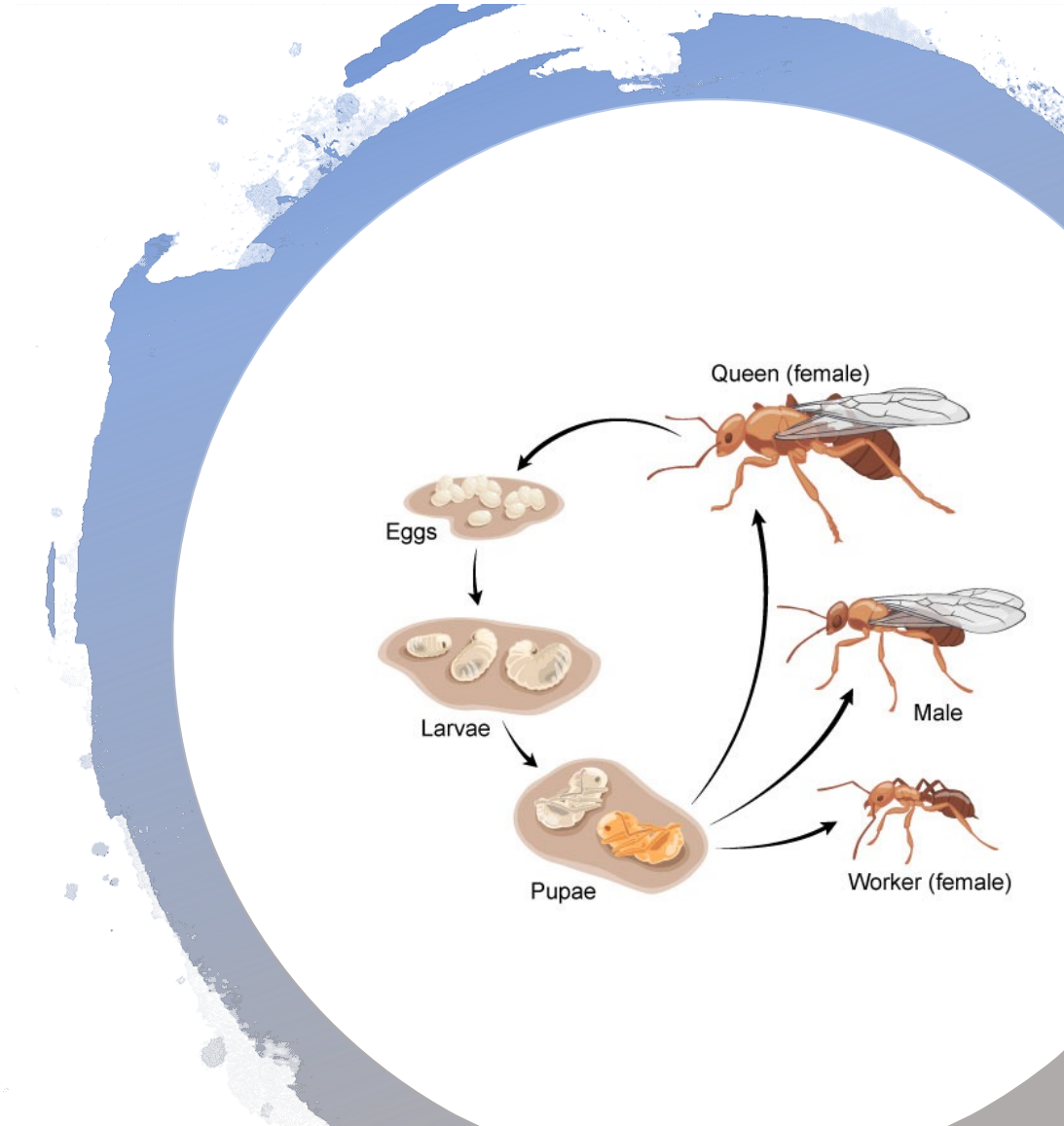


Jin L.C. Guo

M3 (a) – Object State

Image Source: <https://askabiologist.asu.edu/individual-life-cycle>



Objective

- Programming mechanism:
Null references, optional types
- Concepts and Principles:
Object life cycle
- Design techniques:
State Diagram
- Design Patterns and Antipatterns:
NULL OBJECT

Object at Run-time

```
public final class Card
```

```
{
```

```
    private final Rank aRank;
```

```
    private final Suit aSuit;
```

```
}
```

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

```
{CLUBS, DIAMONDS, HEARTS, SPADES}
```

13x4 possible state

Object at Run-time

Abstract State is needed

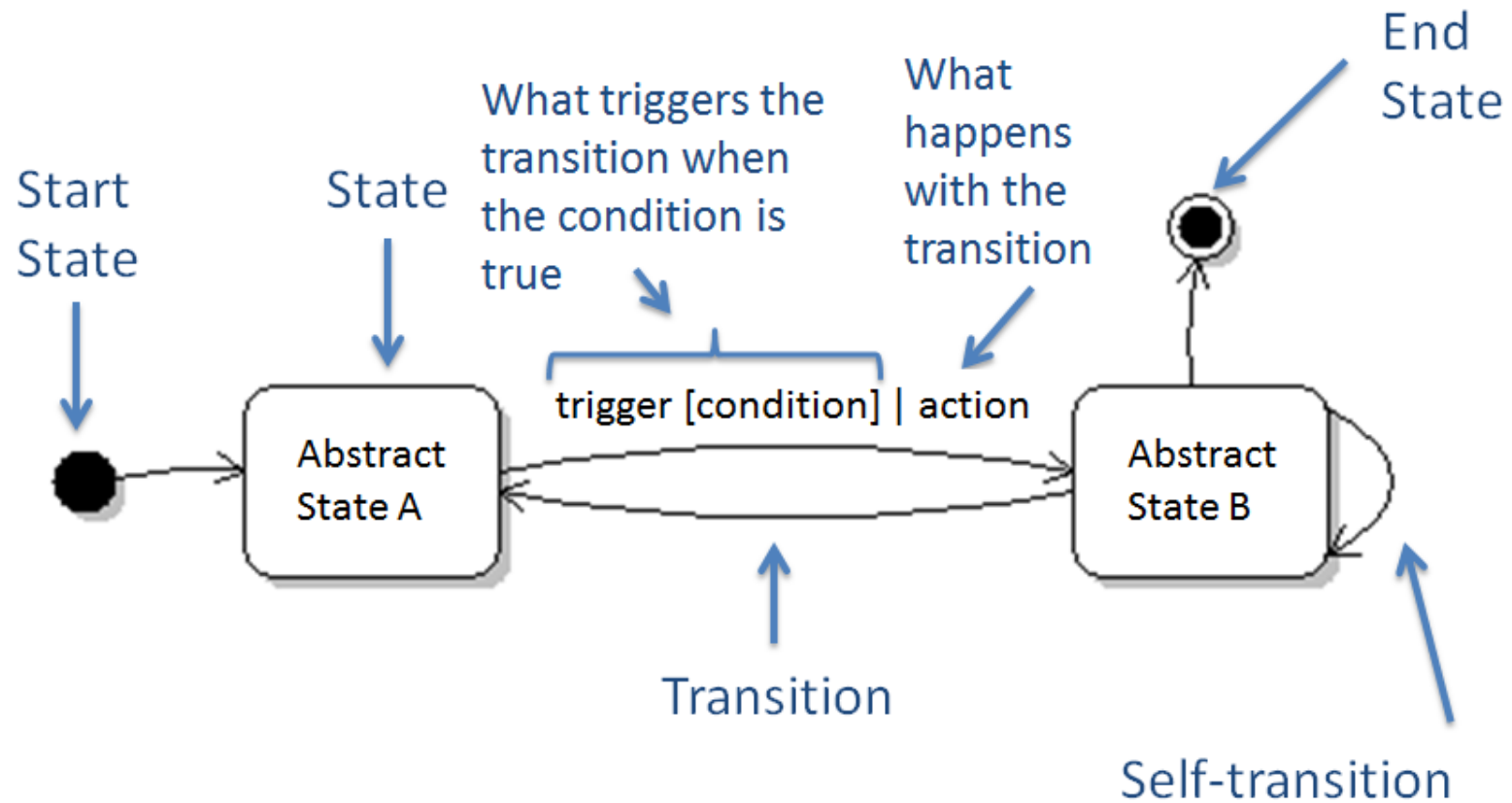
```
public class Student {  
    // Representation of a word in its original form  
    // as in one sentence.  
    private final String firstName;  
}
```

Possible state of the object
 $(2^{31} - 1) \times 2^{16}$!

State Diagram

- Abstract States
- Transitions between states

State Diagram



State diagram of Card

```
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;
    }

    .....
}
```

State diagram of Deck

```
public class Deck
{
    final List<Card> aCard = new ArrayList<>();

    public Deck()
    {
        shuffle();
    }

    public Card draw()
    {
        assert !isEmpty();
        return aCard.remove(aCards.size()-1);
    }

    /**
     * Reinitialize the deck with all 52 cards and shuffles them.
     */
    public void shuffle()
    {
        .....
    }

    .....
}
```


Activity 1: *Sketch* the state diagram of Course

```
public class Course {  
    private String      aID; // e.g. "COMP 303"  
    private boolean     aIsActive;  
    private int         aCap;  
    private List<Student> aEnrollment;  
    private CourseSchedule aSchedule;  
}
```

What might be useful states of a Course object?

How a Course object would transit from one state to another?

Design Constructor

- A constructor should fully initialize the object
 - The class invariant should hold
 - Shouldn't need to call other methods to “finish” initialization

State diagram of Deck

```
public class Deck
{
    final List<Card> aCard = new ArrayList<>();

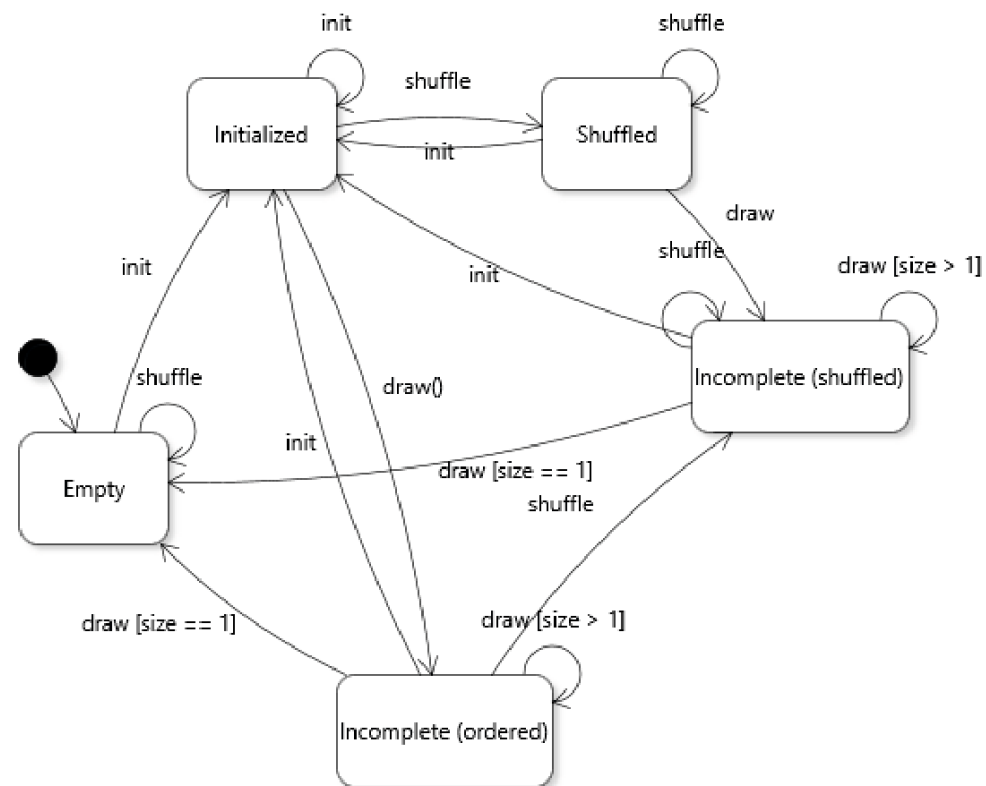
    public Deck()
    {
        shuffle();
    }

    public Card draw()
    {
        assert !isEmpty();
        return aCard.remove(aCards.size()-1);
    }

    /**
     * Reinitialize the deck with all 52 cards and shuffles them.
     */
    public void shuffle()
    {
        .....
    }

    .....
}
```

State diagram of **Deck** *without fully initialization*



Design Field

- Has a value that retains meaning throughout the object's life
- Its state must persist between public method invocations

General Principle

- Minimize the state space of object to what is absolutely necessary
 - It's impossible to put the object in an invalid or useless state
 - There's no unnecessary state information

Objective

- Programming mechanism:

Null references, optional types

- Concepts and Principles:

Object life cycle

- Design techniques:

State Diagram

- Design Patterns and Antipatterns:

NULL OBJECT

Nullability (absence of value)

```
Card card = null;
```

A variable is temporarily un-initialized and will be initialized in a different state.

A variable is incorrectly initialized. The code of initiation is not executed properly.

As a flag that represents the absence of a useful value

Special use.

```
Card.Rank rank = card.getRank();
```

Avoid *null* values when designing classes!

Avoid *null* values when designing classes?

```
public class Course {
```

```
    private String aID;
```

```
    private boolean aIsActive;
```

```
    private int aCap;
```

```
    private List<Student> aEnrollment;
```

It might be a valid state when the class is created but not scheduled.

```
    private CourseSchedule aSchedule;
```

What about Schedule?

```
public Course(String pID, int pCap) {
```

```
    aID = pID;
```

```
    aCap = pCap;
```

```
    aEnrollment = new ArrayList<>();
```

```
    aIsActive = false;
```

```
}
```

Avoid *null* values when designing classes?

- Sometimes it's necessary to model absence of value

Activity 2:

- Discuss your design of the extension of class Card where one instance can also represent a "Joker". (Textbook Chapter 2 - Exercise #4)

Note: Joker is special card with no rank and no suit.

- How did you handle the fields of Rank and Suit for "Joker"?



Image source: https://upload.wikimedia.org/wikipedia/commons/6/6f/Joker_Card_Image.jpg

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

Arbitrary (valid) value for rank and suit?

Add special value for Rank and Suit enums?

Objective

- Programming mechanism:
Null references, optional types
- Concepts and Principles:
Object life cycle
- Design techniques:
State Diagram
- Design Patterns and Antipatterns:
NULL OBJECT

java.util.Optional<T>

- A container object which may or may not contain a non-null value.
- If a value is present, `isPresent()` will return true and `get()` will return the value.

```
public class Card
{
    private Optional<Rank> aRank;
    private Optional<Suit> aSuit;
    private boolean aIsJoker;
```

```
public Card(Rank pRank, Suit pSuit)
{
    assert pRank != null && pSuit != null;
    aRank = Optional.of(pRank);
    aSuit = Optional.of(pSuit);
}
```

```
public Card()
{
    aIsJoker = true;
    aRank = Optional.empty();
    aSuit = Optional.empty();
}
```

What about getter methods?

- Return Optional<T> types
- Up-wrap Optional and return T

Go back to the **Course** class

```
public class Course {  
  
    .....  
  
    public Course(String pID, int pCap) {  
        aID = pID;  
        aCap = pCap;  
        aEnrollment = new ArrayList<>();  
        aIsActive = false;  
        aSchedule = Optional.empty();  
    }  
  
    public void setSchedule(CourseSchedule pSchedule) {  
        aSchedule = Optional.of(pSchedule);  
    }  
  
    public Optional<CourseSchedule> getSchedule(){  
        return aSchedule;  
    }  
}
```

Client code of the **Course** class

```
private static void printSchedule(Course pCourse) {  
    if(pCourse.getSchedule().isPresent()) {  
        CourseSchedule schedule = pCourse.getSchedule().get();  
        System.out.println(schedule);  
    } else {  
        System.out.println("Schedule unavailable.");  
    }  
}
```

Objective

- Programming mechanism:
Null references, optional types
- Concepts and Principles:
Object life cycle
- Design techniques:
State Diagram
- Design Patterns and Antipatterns:

NULL OBJECT

Null Object Pattern

- Use polymorphism to handle absence
- Create a subtype to act as a null version of the class, make it singleton
- *Special Case* Pattern, representing absence, unknown, etc.

```

public class Deck
{
    private final List<Card> aCard = new ArrayList<>();

    private Comparator<Card> aComparator;

    public Card draw()
    {
        assert !isEmpty();
        return aCard.remove(aCards.size()-1);
    }

    public void sort()
    {
        Collections.sort(aCards, aComparator);
    }

    .....
}

```

How to compare the Card objects is unknown when we initialize a Deck object?

```
public class NullComparator implements Comparator<Card>
{
    @Override
    public int compare(Card o1, Card o2) {return 0;}
}
```

Another Example

```
public interface CardSource extends Cloneable
{
    Card draw();

    boolean isEmpty();
}
```

```
public interface CardSource extends Cloneable
{
    public static CardSource NULL = new CardSource() {
        @Override
        public Card draw() {
            assert !isEmpty();
            return null;
        }

        @Override
        public boolean isEmpty() {
            return true;
        }
    };

    Card draw();
    boolean isEmpty();
}
```


Recap - Objective

- Programming mechanism:
Null references, optional types
- Concepts and Principles:
Object life cycle
- Design techniques:
State Diagram
- Design Patterns and Antipatterns:
NULL OBJECT