CS1101 Programming and Problem Solving

Dr. Gina Bai

Spring 2023

Logistics

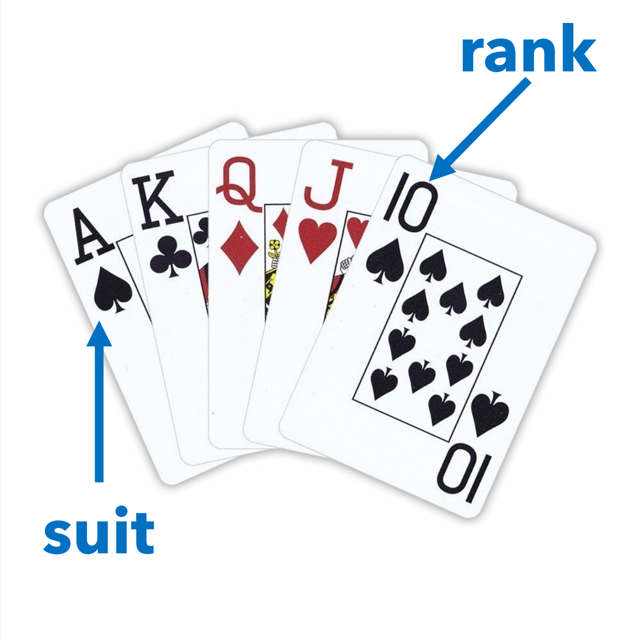
- **ZY-8B** on zyBook > Assignments
  - Due: **Wednesday, April 12**, at 11:59pm
- **ZY-9** on zyBook > Assignments
  - Due: **Wednesday, April 19**, at 11:59pm
- **PA11 – A, B** on zyBook > Chap 11
  - Due: **Thursday, April 20**, at 11:59pm

Start Early!!!

UML Diagrams

zyBook Chap 10.6

“UML Diagram” for Card Class in PA11-A



Class Name
Card
Constructors
+ Card() + Card(rank : int, suit : int)
Instance Variables (Fields)
- rank : int - suit : int
Methods
+ getRank() : String + getSuit() : String + toString() : String + equals(object : Object) : boolean

NOT STANDARD

UML Diagram

<<Java Class>> **Card**
+ SUIT: String[] + RANK: String[] - rank: int - suit: int
+ Card() + Card(int, int) + getRank(): String + getSuit(): String + toString(): String + equals(Object): boolean

- Consists of…
  - ClassName
  - Fields
    - **varName: type**
  - Methods
    - **Constructor(s)**
    - **methodName (<param type>): return type**
    - Note: param names are not included
  - Note: Static fields and methods are indicated by underlining
- Access Modifier
  - **+** or green circle ● : public
  - **-** or red square ◘ : private

Static Fields and Methods

zyBook Chap 9.12

Non-Static vs. Static Fields

- **Non-static** fields
  - A.k.a. **instance variables**
  - Attributes/Properties/Fields of an object
- **Static** fields
  - A.k.a. **class variables**
  - Information **shared** by **all instances** of this class

Recap: An object is an instance of a class

Static Fields

Static field → a field of the class instead of a field of each class object

- Declared and initialized in the class
- Shared by all instances of the class
- **Independent of any class object**
- **Public** static field can be accessed without creating a class object: `<ClassName>.<fieldName>`
  - E.g., Math.PI

```java
public class Robot {
    // Non-static field / instance variables
    private double posX;
    private double posY;
    private int id;

    // Static field / class variable
    public static int nextRobotID = 1;

    public Robot(double posX, double posY) {
        this.posX = posX;
        this.posY = posY;
        id = nextRobotID;

        ++nextRobotID;
    }

    public String toString() {
        return "r" + id + ": (" +
                posX + ", " + posY + ")";
    }
}
```

Example

```java
public class Robot {
    // Non-static field / instance variables
    private double posX;
    private double posY;
    private int id;

    // Static field / class variable
    public static int nextRobotID = 1;

    public Robot(double posX, double posY) {
        this.posX = posX;
        this.posY = posY;
        id = nextRobotID;

        ++nextRobotID;
    }

    public String toString() {
        return "r" + id + ": (" +
               posX + ", " + posY + ")";
    }
}
```

```java
import java.util.Arrays;

public class RobotClient {
    public static void main(String[] args) {
        // Array of Objects - Two-phase initialization
        Robot[] r = new Robot[5];
        for (int i = 0; i < r.length; ++i){
            r[i] = new Robot(i, i);
            System.out.println("Constructed robot " + r[i]);
            System.out.println("The ID of next robot is "
                                   + Robot.nextRobotID);
        }
    }
}
```

r[0] = new Robot (0, 0)

That is,
r[0].posX == 0.0
r[0].posY == 0.0
r[0].id == 1

```
$ javac RobotClient.java
$ java RobotClient
Constructed robot r1: (0.0, 0.0)
The ID of next robot is 2
```

Example

```java
public class Robot {
    // Non-static field / instance variables
    private double posX;
    private double posY;
    private int id;

    // Static field / class variable
    public static int nextRobotID = 1;

    public Robot(double posX, double posY) {
        this.posX = posX;
        this.posY = posY;
        id = nextRobotID;

        ++nextRobotID;
    }

    public String toString() {
        return "r" + id + ": (" +
               posX + ", " + posY + ")";
    }
}
```

```java
import java.util.Arrays;

public class RobotClient {
    public static void main(String[] args) {
        // Array of Objects – Two-phase initialization
        Robot[] r = new Robot[5];
        for (int i = 0; i < r.length; ++i){
            r[i] = new Robot(i, i);
            System.out.println("Constructed robot " + r[i]);
            System.out.println("The ID of next robot is "
                               + Robot.nextRobotID);
        }
    }
}
```

```
$ javac RobotClient.java
$ java RobotClient
Constructed robot r1: (0.0, 0.0)
The ID of next robot is 2
Constructed robot r2: (1.0, 1.0)
The ID of next robot is 3
Constructed robot r3: (2.0, 2.0)
The ID of next robot is 4
Constructed robot r4: (3.0, 3.0)
The ID of next robot is 5
Constructed robot r5: (4.0, 4.0)
The ID of next robot is 6
```

Static Methods vs. Non-Static (Instance) Methods

Static member method → a class method that is **independent of class objects**.

- Typically used to and **can only access** and **mutate** the **private static fields** from outside the class.

```java
public class Robot {
    // Non-static field / instance variables
    private double posX;
    private double posY;
    private int id;

    // Static field / class variable
    private static int nextRobotID = 1;

    public Robot(double posX, double posY) {
        this.posX = posX;
        this.posY = posY;
        id = nextRobotID;

        ++nextRobotID;
    }

    // Non-static method / Instance method
    public int getID() {
        return id;
    }

    // Static method
    public static int getNextRobotID() {
        return nextRobotID;
    }

    public String toString() {
        return "r" + id + ": (" +
                posX + ", " + posY + ")";
    }
}
```

Example

```java
import java.util.Arrays;

public class RobotClient {
    public static void main(String[] args) {
        // Array of Objects - Two-phase initialization
        Robot[] r = new Robot[5];
        for (int i = 0; i < r.length; ++i){
            r[i] = new Robot(i, i);
            System.out.println("Constructed robot #" + r[i].getID());
            System.out.println("The ID of next robot is "
                                  + Robot.getNextRobotID());
        }
    }
}
```

```
$ javac RobotClient.java
$ java RobotClient
Constructed robot #1
The ID of next robot is 2
Constructed robot #2
The ID of next robot is 3
Constructed robot #3
The ID of next robot is 4
Constructed robot #4
The ID of next robot is 5
Constructed robot #5
The ID of next robot is 6
```

```java
public class Robot {
    // Non-static field / instance variables
    private double posX;
    private double posY;
    private int id;

    // Static field / class variable
    private static int nextRobotID = 1;

    public Robot(double posX, double posY) {
        this.posX = posX;
        this.posY = posY;
        id = nextRobotID;

        ++nextRobotID;
    }

    // Non-static method / Instance method
    public int getID() {
        return id;
    }

    // Static method
    public static int getNextRobotID() {
        return nextRobotID;
    }

    public String toString() {
        return "r" + id + ": (" +
                posX + ", " + posY + ")";
    }
}
```

Interacting Classes and Object-Oriented Design

zyBook Chap 9.6

Be Creative and Reasonable

Four Main Principles of OOP

1. **Abstraction**

   To simplify reality and focus only on the properties and external behaviors rather than inner details

2. **Encapsulation**

   Hiding the implementation details (data and the programs that manipulate the data) of an object from the clients of the object

3. Inheritance
4. Polymorphism

Design a Program/Application

- Design Process
  - Determine the classes
  - Determine the responsibilities of each class
  - Determine the interactions and collaborations among the classes

Determine the Classes

- "Just as a **noun** is a person, place, or thing, so is an object."
- Begin by noting the **nouns** in the problem statement.
  - These nouns give us a good starting point for considering possible classes.
  - Not all nouns will become classes
  - Not all classes will correspond to nouns of the problem statement.

Task – Design a Student Class

- How?

Determine Responsibilities of Each Class

- “As the nouns indicate classes, the **verbs** of the problem statement help determine class responsibilities.”
- Consider the following:
  - What service does the class provide?
  - What is each class’s responsibility?
  - What are the actions and behaviors of each class?
  - What attributes/fields?

Brainstorm – Design a Student Class

- Fields for a Student?
  - Name
  - Preferred Name
  - Major
  - …
- Methods for a Student?
  - Generally, getter and setter methods
  - …

Design a Class

- Do not provide any functionality that does not have a clear use
- Limit object creation to the constructor
- Classes should have cohesion
  - The extent to which the code for a class represents a single abstraction
  - Allows for reusability of the class in other programs
- Classes should not have unnecessary dependencies
  - Coupling is the degree to which one part of a program depends on another
  - Related data and behavior should be in the same place (same class)

Brainstorm – Interacting with Student Class

- Objects that Student can interact with?
  - Name
  - Course
  - Dorm
  - Calendar
  - Faculty
  - Hometown
  - Restaurants
  - …

Coding Practice

- Step 1: Complete the TODOs in the classes
- Step 2: Complete the TODOs in the client program
- Step 3: Add more fields and methods to Student Class
- Step 4: Try to come up with objects that Student can interact with, such as Course, Dorm, Calendar, Faculty…