# 实验4

## 实验目的：

* 熟悉数组的使用
* 简单的JavaFX

## 实验要求：

* 下周前五前将工程文档和实验报告打包上传到FTP实验题目：

**一、基本题目**

1． *(Game of Craps)* Write an application that runs 1,000,000 games of craps and answers the following questions:

* a)  How many games are won on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
* b)  How many games are lost on the first roll, second roll, ..., twentieth roll and after the twentieth roll?
* c)  What are the chances of winning at craps? [*Note:* You should discover that craps is one of the fairest casino games. What do you suppose this means?]
* d)  What is the average length of a game of craps?
* e)  Do the chances of winning improve with the length of the game?

2. *(Airline Reservations System)* A small airline has just purchased a computer for its new automated reservations system. You’ve been asked to develop the new system. You’re to write an application to assign seats on each flight of the airline’s only plane (capacity: 10 seats).

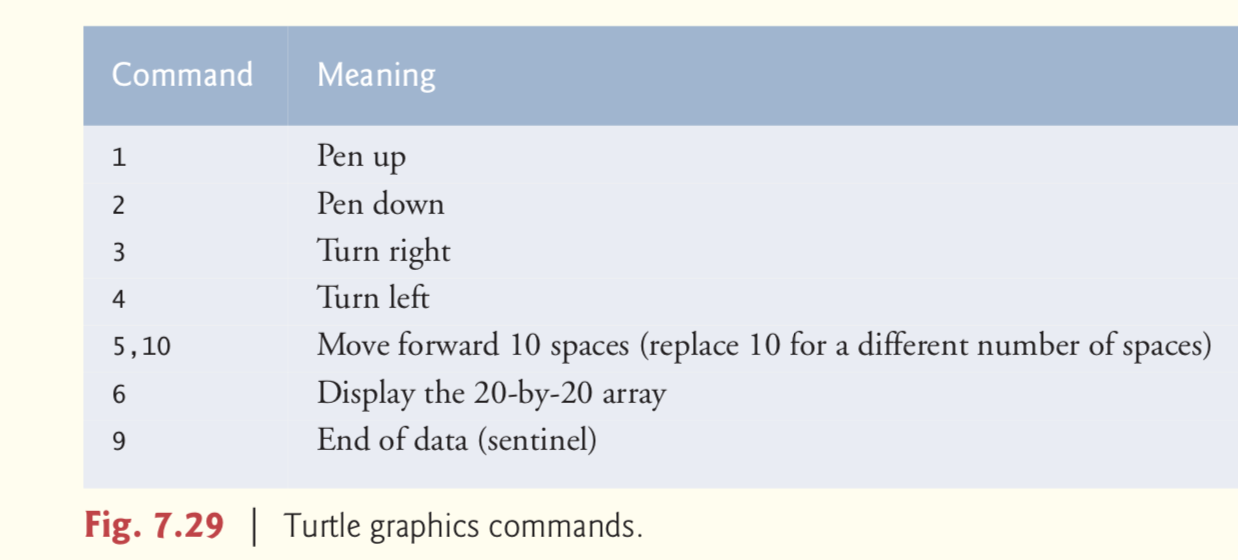
Your application should display the following alternatives: Please type 1 for First Class and Please type 2 for Economy. If the user types 1, your application should assign a seat in the first- class section (seats 1–5). If the user types 2, your application should assign a seat in the economy section (seats 6–10). Your application should then display a boarding pass indicating the person’s seat number and whether it’s in the first-class or economy section of the plane.

Use a one-dimensional array of primitive type boolean to represent the seating chart of the plane. Initialize all the elements of the array to false to indicate that all the seats are empty. As each seat is assigned, set the corresponding element of the array to true to indicate that the seat is no longer available.

Your application should never assign a seat that has already been assigned. When the economy section is full, your application should ask the person if it’s acceptable to be placed in the first-class section (and vice versa). If yes, make the appropriate seat assignment. If no, display the message "Next flight leaves in 3 hours."

3. *(Turtle Graphics)* The Logo language made the concept of *turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a Java application. The turtle holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves, and while the pen is up, the turtle moves about freely without writing anything. In this problem, you’ll simulate the operation of the turtle and create a computerized sketchpad.

Use a 20-by-20 array floor that’s initialized to zeros. Read commands from an array that contains them. Keep track of the current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at position (0, 0) of the floor with its pen up. The set of turtle commands your application must process are shown in Fig. 7.29.



Suppose that the turtle is somewhere near the center of the floor. The following “program”

would draw and display a 12-by-12 square, leaving the pen in the up position:

2

5,12

3

5,12

3

5,12

3

5,12

1

6

9

As the turtle moves with the pen down, set the appropriate elements of array floor to 1s. When the 6 command (display the array) is given, wherever there’s a 1 in the array, display an asterisk or any character you choose. Wherever there’s a 0, display a blank.

Write an application to implement the turtle graphics capabilities discussed here. Write several turtle graphics programs to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

4. *(Enhancing Class Time2)* Modify class Time2 to include a tick method that increments the time stored in a Time2 object by one second. Provide method incrementMinute to increment the minute by one and method incrementHour to increment the hour by one. Write a program that tests the tick method, the incrementMinute method and the incrementHour method to ensure that they work correctly. Be sure to test the following cases:

* a)  incrementing into the next minute,
* b)  incrementing into the next hour and
* c)  incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

5. (Addition App) Create a JavaFX version of the addition program. Use two TextFields to receive the user’s input and a Button to initiate the calculation. Display the results in a Label. Since TextField method getText returns a String, you must convert the String the user enters to an int for use in calculations. Recall that the static method parseInt of class Integer takes a String argument representing an integer and returns the value as an int.

**二、 扩展题目**

1. (7.30 Card Shuffling and Dealing) Modify Fig. 7.11 to deal a five-card poker hand. Then modify class DeckOfCards of Fig. 7.10 to include methods that determine whether a hand contains

1. a pair（一对 5张中2张的数字一样，另外3张歌不相同）
2. two pairs（两对 4张中两两数字一样，余下1张不同）
3. three of a kind(e.g., three jacks 三条 5张中3张的数字一样)
4. four of a kind(e.g.,four aces四条)
5. a flush(i.e.,all five cards of the same suit,同花 五张花色一样)
6. a straight(i.e.,five cards of consecutive face values，顺子 五张数字相连)
7. a full house(i.e., two cards of one face value and three cards of another face value 葫芦3张数字一样，另外2张数字一样)

https://baijiahao.baidu.com/s?id=1694625679352555083&wfr=spider&for=pc



2. (7.31 Card Shuffling and Dealing) Use the methods developed in Exercise 7.30 to write an application that deals two five-card poker hands, evaluates each hand and determines which is better.

3. (7.32 Project: Card Shuffling and Dealing) Modify the application developed in Exercise 7.31 so that it can simulate the dealer. The dealer’s five-card hand is dealt “face down,” so the player cannot see it. The application should then evaluate the dealer’s hand, and, based on the quality of the hand, the dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The application should then reevaluate the dealer’s hand. [Caution: This is a difficult problem!]

4. (Project: Card Shuffling and Dealing) Modify the application developed in Exercise 7.32 so that it can handle the dealer’s hand automatically, but the player is allowed to decide which cards of the player’s hand to replace. The application should then evaluate both hands and determine who wins. Now use this new application to play 20 games against the computer. Who wins more games, you or the computer? Have a friend play 20 games against the computer. Who wins more games? Based on the results of these games, refine your poker-playing application. (This, too, is a difficult problem.) Play 20 more games. Does your modified application play a better game?

5.(Project: Card Shuffling and Dealing) Modify the application of Figs. 7.9 –7.11 to use Face and Suit enum types to represent the faces and suits of the cards. Declare each of these enum types as a public type in its own source-code file. Each Card should have a Face and a Suit instance variable. These should be initialized by the Card constructor. In class DeckOfCards, create an array of Faces that’s initialized with the names of the constants in the Face enum type and an array of Suits that’s initialized with the names of the constants in the Suit enum type. [Note: When you output an enum constant as a String, the name of the constant is displayed.]