

A few words about the quiz

- The mean this time is around 58. To make it meaningful to choose best 3 of 4 quizzes, I'll normalize this quiz by multiplying 1.25 to each mark.
- Most students do not understand what is a loop invariant.
- Fallacies (**wrong** things that are mistaken to be right):
 - ✗ Loop invariant is a statement in the loop.
 - ✗ Loop invariant is a variable used in the loop.
 - ✗ Loop invariant means explaining the loop body.
 - ✗ Loop invariant is the loop condition.

0911B L15-1

Fact about loop invariants

- Loop invariants are the status of the program when a loop starts executing that are useful for the understanding of the loop.
- They mostly concerns the variables that are modified in the loop.
- The loop invariants **must** be able to tell what is the value of these variables, or at least what are their relationship.
- Loop invariants **cannot** be extracted from the program.
- The loop condition is at best a very weak loop invariant—usually we cannot understand the loop by looking at the loop condition.
- You have to understand a loop quite a bit before you know the loop invariant.

0911B L15-2

Example

Loop invariant of the loop in the following code:

```
int n = in.readInt();
int num_digits = 0;
while (n > 0) {
    ++ num_digits;
    n /= 10;
}
```

Two variables are modified: num_digits, n. The loop invariant must specify their value at the beginning of the while statement.

Note that num_digits starts with 0 and is incremented every time, so num_digits stores the number of times the loop had been entered—loop invariant part 1.

How about n? It is divided by 10 each time the loop is entered, so n stores the result dividing the input by 10 for num_digits times—loop invariant part 2 (more important).

0911B L15-3

The horrible world without design

Without designing a program first before writing, a program becomes a complete mess.

E.g., Printing the first week of a month...

```
// day-of-week already in int variable day_of_week
for (int i = 1; i <= 7; ++i) {

    Fmt.printf("%3d", i);

}
System.out.println();
```

0911B L15-4

Designing with loop invariants

Question 1: What should be done in each iteration?

Answer 1: Print out one field, whether it is blank or a day.

Question 2: What is the loop invariant?

Answer 2: A loop variable *i* is set to the field number to be printed. Every day before it is already printed, and all other days are not yet printed. I.e., anything on day *d* must be done when *i*=*d*.

```
// day-of-week already in int variable day_of_week
for (int i = 1; i <= 7; ++i) {
    if (i <= day_of_week)
        System.out.print(" ");
    else
        Fmt.printf("%2d ", i - day_of_week);
}
System.out.println();
```

0911B L15-5

Revision

- We have already learnt a lot of way to control when a statement is executed.
- The simple statement is always executed only once.
- The if/else statement and the switch statement select one thing to do among two or more things.
- The while and for statements do one thing many times.
- Compound statement allows you to treat things as one big thing.
- Method allows you to define a new "thing" that can be done by a simple statement (method call).

0911B L15-6

The need for better data structure constructs

- The things we have learnt do not yet allow us to do really complicated things. One piece of the puzzle is missing.
- At the beginning of the course, we learnt that Java provides eight primitive types: byte, short, int, long, float, double, char, boolean.
- With only primitive types, we can only store a fixed number of things. That is, when we compile the program, we must know how many int, double, boolean, char we want.
- When we don't know how many things we need until the program actually runs, we need better data types—objects.
- The first type of Java objects we learn is "array".

0911B L15-7

Example problem

Read a number n from the input. Then read n double-type numbers, and find their median.

- Unlike finding averages and variance (i.e., the program in the quiz), there is no way to find the median without first storing all the input data.
- The input data consists of n numbers, which itself is unknown until the program is executed. This cannot be done with anything we already know.
- What we need is a statement that creates a lot of double, and a way for us to refer to one of these double's.
- **Arrays** allow you to create a lot of *elements of the same type*. You can access them through an **object reference**.

0911B L15-8

Making arrays

To create an array, we use the **new** operator. The expression is of the following form:

```
new type[size]
```

The array contains *size* elements, which can be any non-negative number. If you give it a negative number, it causes a run-time error.

The elements of the allocated array is numbered from 0 to *size*-1.

Initially, each element stores something like 0, 0.0, false, '\u0000' and "".

For example, we can write `new double[200]` and at once we have 200 double values allocated, numbered 0 to 199, each contains 0.0.

There is no need to deallocate arrays: Java automatically deallocates them once you cannot use it any more.

0911B L15-9

Using the array allocated

- Note that the **new** expression does not make a variable. The allocated array does not have a name.
 - The expression `new double[200]` returns a **reference** that you can use to access the array.
 - Most likely, you'll store the reference in a variable, which must be of type
- ```
type[]
```
- So the whole statement that makes an array and stores the reference in a new variable looks like this:

```
double[] arr = new double[200];
```

0911B L15-10

### Using elements of arrays

- As we have said, the elements of an array of size  $n$  is numbered from 0 to  $n - 1$ .
- With an array reference (e.g. `arr`), we can access the  $i$ -th element by using the **array reference** operator (`[]`), like `arr[i]`.
- If you access an element outside the range from 0 to  $n - 1$ , you get a run-time error.
- You can use an expression like `arr[i]` just like a variable:
  - Put it on the left of an assignment expression or use increment or decrement operators to change its value.
  - Write the expression `arr[i]` in other expressions to get its value.

0911B L15-11

### Example

For the problem we describe at the beginning of the lecture, we create an array to store all the input and use the reference named `data` to refer to it:

```
import chapman.io.*;
public class FindMedian {
 public static void main(String[] args) {
 StdIn in = new StdIn();
 System.out.print("Number of elements: ");
 int size = in.readInt();
 double[] data = new double[size];
 System.out.println("Input the numbers separated by <Enter>:");
 for (int i = 0; i < size; ++i)
 data[i] = in.readDouble();
 System.out.print("FIXME: should find and print the median of\n ");
 for (int i = 0; i < size; ++i)
 System.out.print(data[i] + " ");
 System.out.println();
 }
}
```

0911B L15-12

## Standard methods for arrays

One way to find the median is first to sort the array and then to find the middle element. There is standard methods to sort an array. We can also do other things with standard methods. E.g.,

- Given an array `arr`, we can sort it from smallest to largest with `java.util.Arrays.sort(arr);`
- Given an array `arr`, we can assign a value `val` into every element of the array by `java.util.Arrays.fill(arr, val);`
- Given two arrays `arr1` and `arr2`, we can check whether they are equal by `java.util.Arrays.equals(arr1, arr2);`
- Given a sorted array `arr`, we can look for a value `val` in it using `java.util.Arrays.binarySearch(arr, val);` which returns the index of the found element, or a negative number if it is not there.

0911B L15-13

## Finding median

Again, we can use `import` so that the methods are easier to call.

```
/* Find the Median of numbers read from the input */
import chapman.io.*;
import java.util.*;
public class FindMedian {
 public static void main(String[] args) {
 StdIn in = new StdIn();
 System.out.print("Number of elements: ");
 int size = in.readInt();
 double[] data = new double[size];
 System.out.println("Input the numbers separated by <Enter>:");
 for (int i = 0; i < size; ++i)
 data[i] = in.readDouble();
 Arrays.sort(data);
 System.out.print("Median is ");
 if (size % 2 == 1)
 System.out.println(data[size/2]);
 else
 System.out.println((data[size/2-1]+data[size/2])/2);
 }
}
```

0911B L15-14

## Array initializer

Sometimes we want an array to store some known, fixed data. In this case, we use an **initializer** to explicitly give initial values to the array, instead of using `new`.

E.g.: If we want to have all month names in an array, we write:

```
final String[] month_names = {
 "January", "February", "March", "April", "May", "June",
 "July", "August", "September", "October", "November", "December"
};
```

Then `month_names[2]` gives you "March", etc. (Note: programmers count from 0, not 1!)

0911B L15-15

## Example: printing a random deck of playing card

- First, we want to make a deck. It must contain all the cards from Club 2 to Spade A. This is simple: we make a loop to create all the cards of a suit. To create all the cards of a suit, we use a loop to create all the numbers of that suit.
- Using the techniques in the last page, we can make the names of the suits and the values of the cards very easily.
- Then we need to shuffle it. We do it by randomly picking a card from 0 to 51 to put into position 51, then randomly picking a card from 0 to 50 to put into position 50, and so on, until we've chosen a card to place at position 1. (Question: Why we don't need to choose a card to place at position 0?)
- Finally, we print out the deck.

0911B L15-16

## Program

```
import chapman.io.*;
public class RandomDeck {
 final static String[] suit = { "Spade", "Heart", "Diamond", "Club" };
 final static String[] value = { "2", "3", "4", "5", "6", "7", "8",
 "9", "10", "J", "Q", "K", "A" };
 static String[] cards = new String[52];
 public static void main(String[] args) {
 int card_no = 0;
 for (int i = 0; i < 4; ++i) // Make the cards
 for (int j = 0; j < 13; ++j)
 cards[card_no++] = suit[i] + ' ' + value[j];
 for (int i = 51; i >= 1; --i) { // Shuffle it
 int rand_card = (int)(Math.random() * (i+1)); // Chooses a card
 String t = cards[rand_card]; // Move it to card i
 cards[rand_card] = cards[i];
 cards[i] = t;
 }
 for (int i = 0; i < 52; ++i) { // Print them out
 Fmt.printf("%-15s", cards[i]);
 if (i % 4 == 3) // Print out a new line every 4 cards
 System.out.println();
 }
 }
}
```

0911B L15-17

## Multi-dimensional arrays

- We can make arrays of array, just like arrays of `int` or `char`. This is usually used to make multi-dimensional arrays.
- To create a 2-D array of `int` of size  $m \times n$ , we use the new operator like `new int[m][n]`.
- The variable that holds the generated reference is of type `int[][]`.
- So, to create an array and store it into a reference variable:  
`int[][] a = new int[100][100];`
- Again you can use initializers:  
`int[][] a = { { 2, 3 }, { 4, 5 } };`

0911B L15-18

**Small example: make an initial board of Reverse-It**

The board looks like the follows:

|  |  |  |   |   |  |  |  |  |  |
|--|--|--|---|---|--|--|--|--|--|
|  |  |  |   |   |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |
|  |  |  | ● | ○ |  |  |  |  |  |
|  |  |  | ○ | ● |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |
|  |  |  |   |   |  |  |  |  |  |

We will store it in a two-dimensional array of characters. We use different characters to represent an empty cell, a cell occupied by a white piece, and a cell occupied by a black piece.

0911B L15-19

**Program**

```
// Make a reverse-it board
public class ReverseIt {
 final static char UNFILLED = ' ';
 final static char WHITE = 'O';
 final static char BLACK = '●';
 final static int BOARD_SIZE = 8;
 public static void main(String[] args) {
 char[][] board = new char[BOARD_SIZE][BOARD_SIZE];
 for (int i = 0; i < BOARD_SIZE; ++i)
 for (int j = 0; j < BOARD_SIZE; ++j)
 board[i][j] = UNFILLED;
 board[3][4] = board[4][3] = WHITE;
 board[3][3] = board[4][4] = BLACK;
 printBoard(board);
 }
 public static void printBoard(char[][] b) {
 for (int i = 0; i < BOARD_SIZE; ++i) {
 for (int j = 0; j < BOARD_SIZE; ++j)
 System.out.print(b[i][j]+" ");
 System.out.println();
 }
 }
}
```

0911B L15-20