Shuffling over array elements

```java
for (int i = 0; i < A.length; ++i) {
    // choose j randomly
    int j = (int) (Math.random() * A.length);
    // swap
    int tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
}
```

• How to swap values of two variables without `tmp`?
• However, this naive algorithm is biased.¹

¹See https://blog.codinghorror.com/the-danger-of-naivete/
Exercise

Deck of Cards

Write a program which picks first 5 cards at random from a deck of 52 cards.

- 4 suits: Spade, Heart, Diamond, Club
- 13 ranks: 3, ..., 10, J, Q, K, A, 2
- Label 52 cards by 0, 1, ..., 51
- Shuffle the numbers
- Deal the first 5 cards
String[] suits = {"Spade", "Heart", "Diamond", "Club"};
String[] ranks = {
  "3", "4", "5", "6", "7",
  "8", "9", "10", "J", "Q", "K",
  "A", "2"};

int size = 52;
int[] deck = new int[size];
for (int i = 0; i < deck.length; i++)
  deck[i] = i;

// shuffle over deck; correct version
for (int i = 0; i < size - 1; i++) {
  int j = (int) (Math.random() * (size - i)) + i;
  int z = deck[i];
  deck[i] = deck[j];
  deck[j] = z;
}

for (int i = 0; i < 5; i++) {
  String suit = suits[deck[i] / 13];
  String rank = ranks[deck[i] % 13];
  System.out.printf("%8s%3s\n", suit, rank);
}
...
Cloning Arrays

- In practice, one might duplicate an array for some reason.
- One could attempt to use the assignment statement (=), for example,

```
... T[] A = {...}; // assume A is an array
T[] B = A; // shallow copy; you don’t have a new array
...
```

- However, this is impossible to make two distinct arrays.
- Recall that the array variables are simply references to the arrays in the heap.
• Moreover, all the reference variables share this property!
• For example,
• Use a loop to copy individual elements one by one.

```java
int[] A = {2, 1, 3, 5, 10};
int[] B = new int[A.length];
// deep copy
for (int i = 0; i < A.length; ++i) {
    B[i] = A[i];
}
```

• Alternatively, you may use the `arraycopy` method in the `System` class.

```java
int[] A = {2, 1, 3, 5, 10};
int[] B = new int[A.length];
System.arraycopy(A, 0, B, 0, A.length);
```
for-each Loops

• A for-each loop is designed to iterate over a collection of objects, such as arrays and other data structures, in strictly sequential fashion, from start to finish.

• For example,

```
... T[] A = {...}; // assume some T-type array
for (T element: A) {
    // body
}
... 
```

• Note that the type $T$ should be compatible to the element type of $A$.

\(^2\)Beginning with JDK5. Now we have JDK9.
Example

```
... int[] A = {1, 2, 3};
int sum = 0;
for (int i = 0; i < A.length; ++i)
    sum += A[i];
...
```

- Not only is the syntax streamlined, but it also prevents boundary errors.

```
... int[] A = {1, 2, 3};
int sum = 0;
for (int x: A)
    sum += x;
...```
A data structure is a particular way of organizing data in a program so that it can be used efficiently.

Data structures can implement one or more particular abstract data types (ADT), which specify the operations that can be performed on a data structure and the computational complexity of those operations.

In comparison, a data structure is a concrete implementation of the specification provided by some ADT.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks.³

³See http://bigocheatsheet.com/.
Common Operations on Data

- A specific data structure is chosen in one problem.
- Then the operations are implemented accordingly.
- The **Arrays** class contains useful methods for common array operations such as **sorting** and **searching**.
- For example,

```java
import java.util.Arrays;
...
int[] A = {5, 2, 8};
Arrays.sort(A); // sort the whole array

char[] B = {'A', 'r', 't', 'h', 'u', 'r'};
Arrays.sort(B, 1, 3); // sort the array partially
...
```
Selection Sort

```java
// selection sort
for (int i = 0; i < A.length; i++) {
    int k = i; // the position of min starting from i
    for (int j = i + 1; j < A.length; j++) {
        if (A[k] > A[j])
            k = j;
    }
    // swap(A[i], A[k])
    int tmp = A[k];
    A[k] = A[i];
    A[i] = tmp;
}
```

- Time complexity: \(O(n^2)\)
- You can find more sorting algorithms.\(^4\)

\(^4\)See http://visualgo.net/.
Linear Search

Write a program which searches for the index associated with the key.

- For convenience, assume that there is no duplicate key.
- The linear search approach compares the key with each element in the array sequentially.
// assume A is an array
// linear search
for (int i = 0; i < A.length; i++) {
    if (A[i] == key) {
        System.out.printf("%3d", i);
    }
}

• Time complexity: \( O(n) \)
### Alternative: Binary Search

<table>
<thead>
<tr>
<th>Low</th>
<th>Mid</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>1</td>
<td>[2]</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>45</td>
</tr>
<tr>
<td>50</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td>66</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td>79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Time complexity:** $O(\log n)$
- **Overall time complexity (sorting + searching):** still $O(\log n)$?
...  
  int index = -1; // why?
  int high = A.length - 1, low = 0, mid;
  while (high > low) {
      mid = (high + low) / 2;
      if (A[mid] == key) {
          index = mid;
          break;
      } else if (A[mid] > key)  
          high = mid - 1;
      else  
          low = mid + 1;
  }

  if (index > -1)
      System.out.printf("%d: %d\n", key, index);
  else
      System.out.printf("%d: does not exist\n", key);

...
Beyond 1-Dimensional Arrays

- 2D or high-dimensional arrays are widely used.
  - For example, a colorful image is represented by three 2D arrays (R, G, B).
- We can create a 2D T-type array with 4 rows and 3 columns as follows:

```java
...  
int rowSize = 4; // row size  
int colSize = 3; // column size  
T[][] x = new T[rowSize][colSize];  
...
```
Case (c) shows that we can create a 2D array by enumeration.
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
\[ \text{int[][] A = \{\{1, 2, 3\}, \{4, 5\}, \{6\}\};} \]

// conventional for loop
for (int i = 0; i < A.length; i++) {
    for (int j = 0; j < A[i].length; j++)
        System.out.printf("%2d", A[i][j]);
    System.out.println();
}

// for–each loop
for (int[] B : A) {
    for (int item : B)
        System.out.printf("%2d", item);
    System.out.println();
}

...
Exercise: Matrix Multiplication

Write a program which determines $C = A \times B$ for the input matrices $A_{m \times n}$ and $B_{n \times q}$ for $m, n, q \in \mathbb{N}$.

• You may use the formula

\[
c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}
\]

where $a_{ik}, \ i = 1, 2, \ldots, m$ is a shorthand for $A$ and $b_{kj}, \ j = 1, 2, \ldots, q$ for $B$.

• Time complexity: $O(n^3)$ (Why?)
class Lecture6 {
  "Methods"
}

// keywords:
return
Methods

- Methods can be used to define reusable code, and organize and simplify code.
- The idea of function originates from math, that is,
  \[ y = f(x), \]
  where \( x \) is the input parameter and \( y \) is the function value.
- In computer science, each input parameter should be declared with a specific type, and a function should be assigned with a return type.

---

\(^6\) Recall the multivariate functions. The input can be a vector, say the position vector \((x, y, z)\).

\(^7\) Aka procedures and functions.
Example: max

Define a method:

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

Invoke a method:

```java
int z = max(x, y);
```
• The modifier could be **static** and **public** (for now).
• The returnType could be primitive types and reference types.
  • If the method does not return any value, then the return type is **void**.
• The listOfParameters is the input of the method, separated by commas if there are multiple items.
  • Note that a method could have no input.\(^8\)
• The method name and the parameter list together are called the **method signature**.\(^9\)

\(^8\)For example, **Math.random()**.
\(^9\)Method overloading depends this. We will see it soon.
More Observations

- There are alternatives to the method `max()`:

```java
public static int max(int x, int y) {
    if (x > y) {
        return x;
    } else {
        return y;
    }
}
```

```java
public static int max(int x, int y) {
    return x > y ? x : y;
}
```
“All roads lead to Rome.”
– Anonymous

“但如你根本並無招式，敵人如何來破你的招式？”
– 風清揚，笑傲江湖。第十回。傳劍
The return Statement

- The `return` statement is the end point of the method.
- A `callee` is a method invoked by a `caller`.
- The callee returns to the caller if the callee
  - completes all the statements (w/o a `return` statement, say `main()`);
  - reaches a `return` statement;
  - throws an `exception` (introduced later).
- As you can see, the `return` statement is not necessarily at the bottom of the method.\(^\text{10}\)
- Once one defines the return type (except `void`), the method `should` guarantee to return a value or an object of that type.

\(^\text{10}\)Thanks to a lively discussion on November 22, 2015.
Bad Examples

```java
... public static int fun1() {
    while (true);
    return 0; // unreachable code
}

public static int fun2(int x) {
    if (x > 0) {
        return x;
    }
    // what if x < 0?
}
...`
```