Lecture 5

- Understand concepts of loops
- Different types of loops
  - while loop
  - do-while loop
  - for loop

Motivation

- Very often a program would repeat the same set of procedures several times
  - To compute the grades for different students
  - To move the car continuously
  - To create a moving sequence of images
- Loops allow a block of code to be executed repeatedly

Introduction to Loops

- Three types of loops
  - while loop
  - do-while loop
  - for loop
- Nested Loops
  - Similar to if statements, loops can also be nested

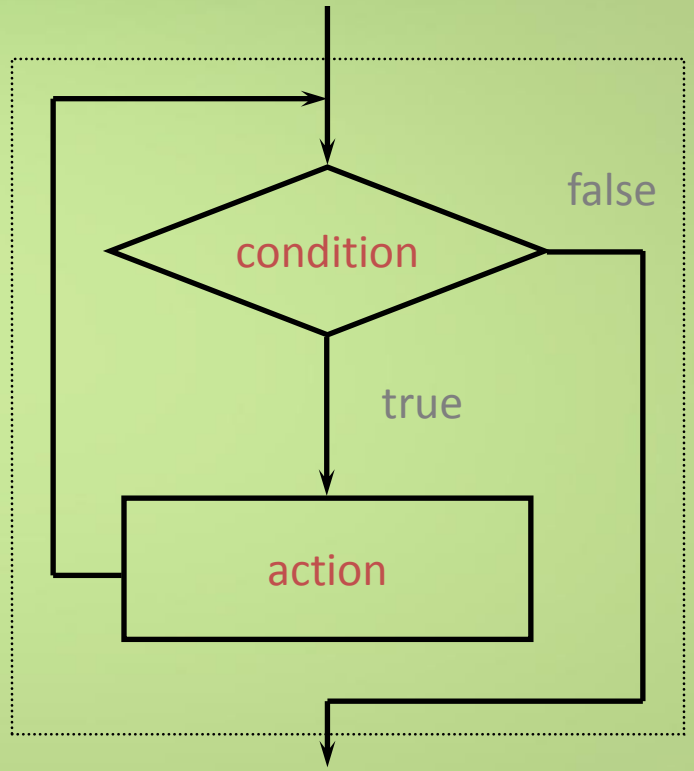
while Loop

- Syntax

  while (condition) {

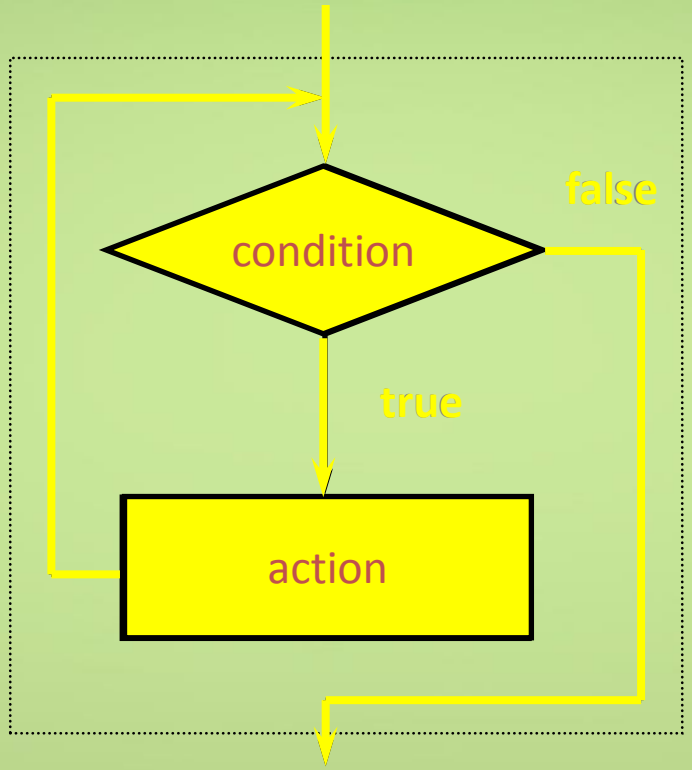
  // action ;

  }
- How does it work?
  - if condition is true then execute action
  - repeat action until condition becomes false
- Action can be a group of statements or a single statement.



A Loop Statement

```
while ( condition )
{
    action ;
}
```



Compute n!

- **n!** (n factorial) is defined as the product of all the integers from 0 to n:

  n! = 1*2*3*…*n (or n! = (n-1)!*n) and 0! = 1

- For example, 5! = 1*2*3*4*5 = 120
- Algorithm for computing n!
  - Initialize the intermediate result **t** to 1 (or 0!) and a **counter** to 1.
  - As long as **counter** is less than or equal n, repeat the computation of **t*counter** and increase **counter** by one.
    - 1! = 0!*1 = t*1 (set **t** to 1, which is 0!)
    - 2! = 1!*2 = **t***2 (update t to this new value, which is 2!)
    - 3! = 2!*3 = **t***3 (update t to this new value, which is 3!)
    - 4! = 3!*4 = **t***4 (update t to this new value, which is 4!)
    - 5! = 4!*5 = **t***5 (update t to this new value, which is 5!)

Compute n! (while loop)

Implement n! using a while loop

- Initialize the intermediate result **t** to 1 (or 0!) and a **counter** to 1
- As long as **counter** is less than or equal n, repeat the computation of **t*counter** and increase **counter** by one.

```
public static int factorial(int number) {
  int t = 1;     // initialize t to 1
  int counter = 1; // initialize counter to 1

  while(counter <= number){
    t = t * counter;
    counter = counter + 1;
  }
  return t;
}
```

Compute 2^n

- 2^n is 2 raised to the n-th power:

  $2^n = 2^{n-1} * 2$

- For example, $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$ …
- Algorithm for computing 2^n
  - Initialize the intermediate result **t** to 1 (or 2^0) and **counter** to 1.
  - As long as **counter** is less than or equal to n, repeat the computation of **t***2 and update **counter**.
    - $2^0 = 1 = t$ (set t to 1, which is 2^0)
    - $2^1 = 2^0 * 2 = t*2$ (update t to this new value, which is 2^1)
    - $2^2 = 2^1 * 2 = t*2$ (update t to this new value, which is 2^2)
    - $2^3 = 2^2 * 2 = t*2$ (update t to this new value, which is 2^3)
    - $2^4 = 2^3 * 2 = t*2$ (update t to this new value, which is 2^4)

Compute 2^n (while loop)

Implement 2^n using a while loop

- Initialize the intermediate result **t** to 1 (or 2^0) and **counter** to 1.
- As long as **counter** is less than or equal to n, repeat the computation of **t***2 and update **counter**.

```
public static int powerTwo(int number) {
  int t = 1;     // initialize t to 1
  int counter = 1; // initialize counter to 1

  while(counter <= number) {
    t = t * 2;
    counter = counter + 1;
  }
  return t;
}
```

How to compute m^n ?

Class or static method

- Class (or static) methods are declared using the static modifier.
  For example:
  ```
  public static int factorial(int number)
  public static int powerTwo (int number)
  ```
- Class methods can be invoked without the need for creating an instance of the class. They can be invoked outside the class with the class name:
  - ClassName.staticMethodName(parameters);
- Instance methods can access instance variable and methods as well as class variable and methods directly.
- Class methods can access class variables methods directly but **not** instance variables and instance methods – they must use an object reference.

Increment and Decrement Operators

- Java has special operators for incrementing (++) and decrementing (--) an integer by one.
- The ++ operator functions as follows:
  - ++a increments the value of a by one and the incremented value is used in the expression.
  - a++ uses the initial value of a in the expression and increments afterwards.

Increment and decrement operators

Operator	Name	Description
++a {y = ++a;}	Pre-increment	Increase **a** by 1 and then use the value of **a** in the assignment {a = a + 1; y = a;}
a++ {y = a++;}	Post-increment	Use the initial value of **a** in the assignment and then increase a by 1 {y = a; a = a + 1;}
--a {y = --a;}	Pre-decrement	Decrease **a** by 1 and then use the value of **a** in the assignment {a = a - 1; y = a;}
a-- {y = a--;}	Post-decrement	Use the initial value of **a** in the assignment then decrease **a** by 1 {y = a; a = a - 1;}

Example

```
//   Prefix and Postfix Increment operators
public void testPrePost ( )  {
      int a;
      int y;
      a = 4;
      IO.outputln("value of a:     " + a  );
      y = a++ + 5;
      IO.outputln ("value of y:    " + y );
      IO.outputln ("new value of a:  " +  a );

      a = 4;
      IO.outputln ("value of a:     " + a );
      y = ++a + 5;
      IO.outputln ("value of y:    " + y );
      IO.outputln ("new value of a:  " +  a  );
}
```

```
/*
Results:
value of a:         4
value of y:         9
new value of a:   5

value of a:         4
value of y:        10
new value of a:   5
*/
```

Shortcut Assignments

- Java has a set of shortcut operators for applying an operation to a variable and then assigning the result back to that variable.
- Shortcut assignments :

	shortcut	same as
*=	`a *= b;`	`a = a*b;`
/=	`a /= b;`	`a = a/b;`
+=	`a += b;`	`a = a+b;`
-=	`a -= b;`	`a = a-b;`
%=	`a %= b;`	`a = a%b;`

Shortcut Assignments

Examples

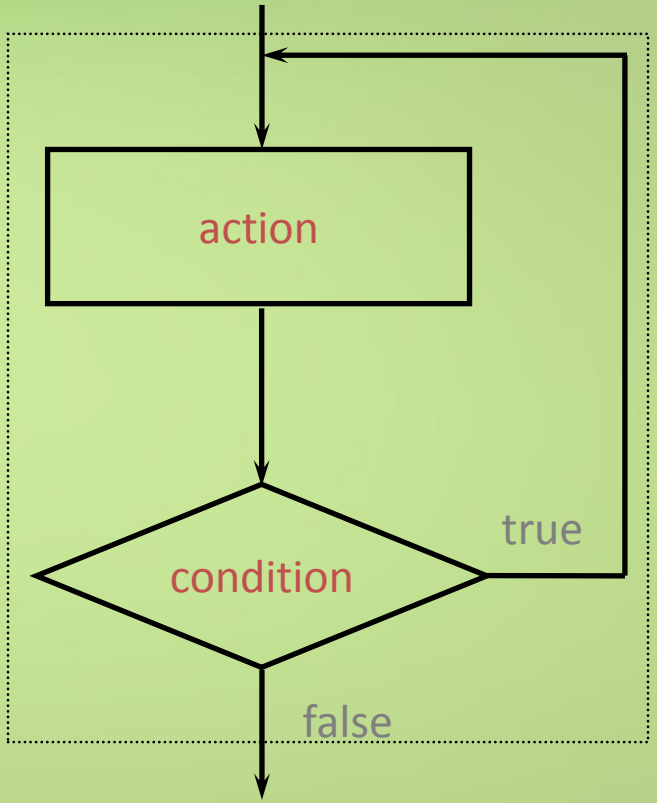
```
int i = 3;
i += 4;                          // i = i + 4
IO.outputln("i = " + i );  // i is now 7

double a = 3.2;
a *= 2.0;                        // a = a * 2.0
IO.outputln("a = " + a);   // a is now 6.4

int b = 15;
b %= 10;                         // b = b % 10
IO.outputln("b = " + b );  // b is now 5
```

do-while Loop

- **Syntax**

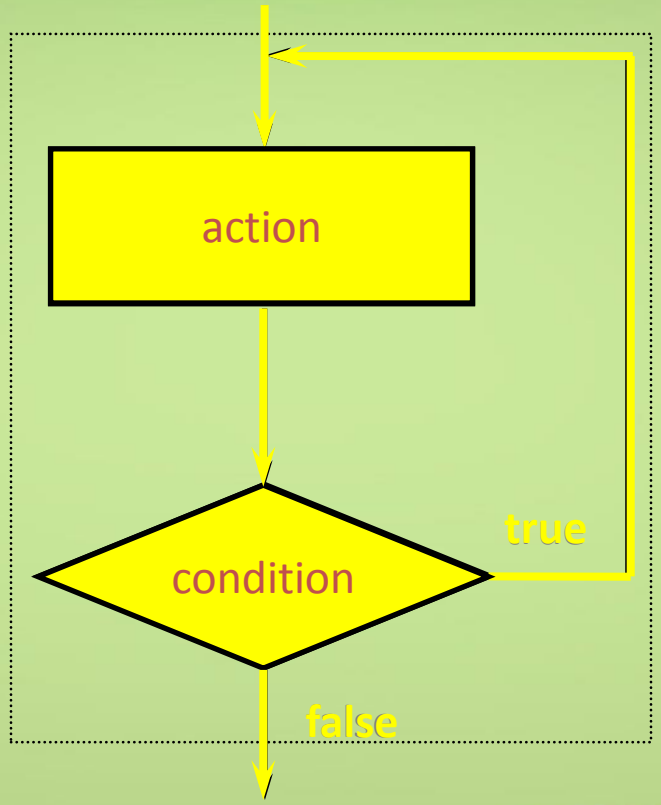
```
do {
    action;
} while (condition);
```

- **How does it work?**
  - execute action
  - if condition is true then execute action again
  - repeat this process until condition evaluates to false



do-while Loop

```
do {
    action;
} while (condition);
```



Compute n! (do-while loop)

Implement n! using a do-while loop

- Initialize the intermediate result **t** to 1 (or 0!) and a **counter** to 1
- Compute **t*counter** and increment **counter** by 1
- As long as **counter** is less than or equal to n, repeat the computation of **t*counter** and update **counter**

```
public static int factorial(int number) {
  int t = 1, counter = 1;

  do {
    t *= counter;  // t = t * counter
    counter += 1;  //counter = counter + 1
  } while(counter <= number); //don't forget the ';'
  return t;
}
```

Compute 2^n (do -while loop)

Implement 2^n using a do-while loop

- Initialize the intermediate result **t** to 1 (or 2^0) and **counter** to 1.
- Compute **t*2** and increment **counter** by 1
- As long as **counter** is less than or equal to n, repeat the computation of **t***2 and update **counter**.

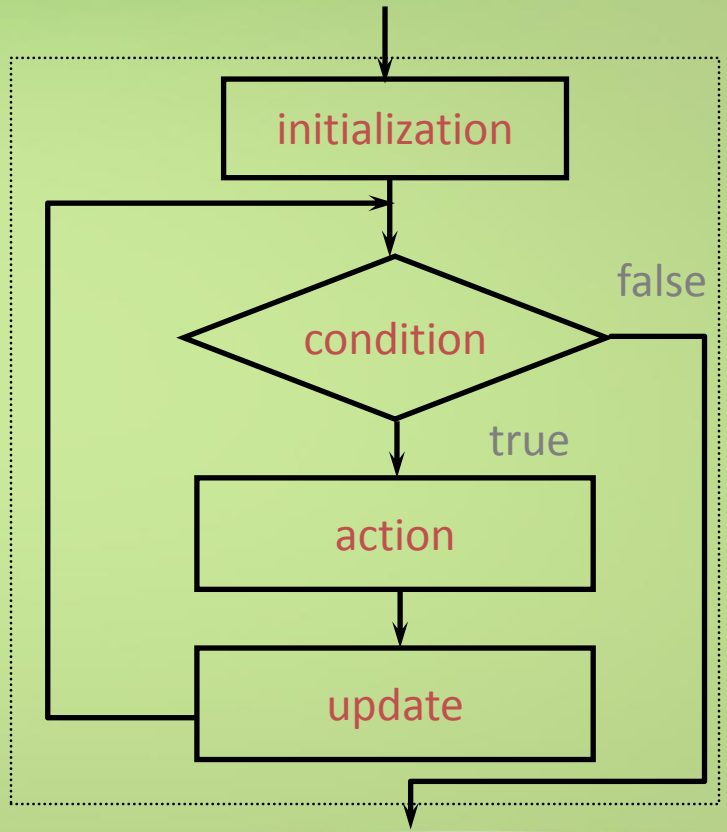
```
public static int powerTwo(int number) {
  int t = 1, counter = 1;

  do {
    t *= 2;                    // t = t*2
    counter += 1;              //counter = counter+ 1
  } while (counter <= number); //don't forget the ';'
  return t;
}
```

for Loop

- **Syntax:**

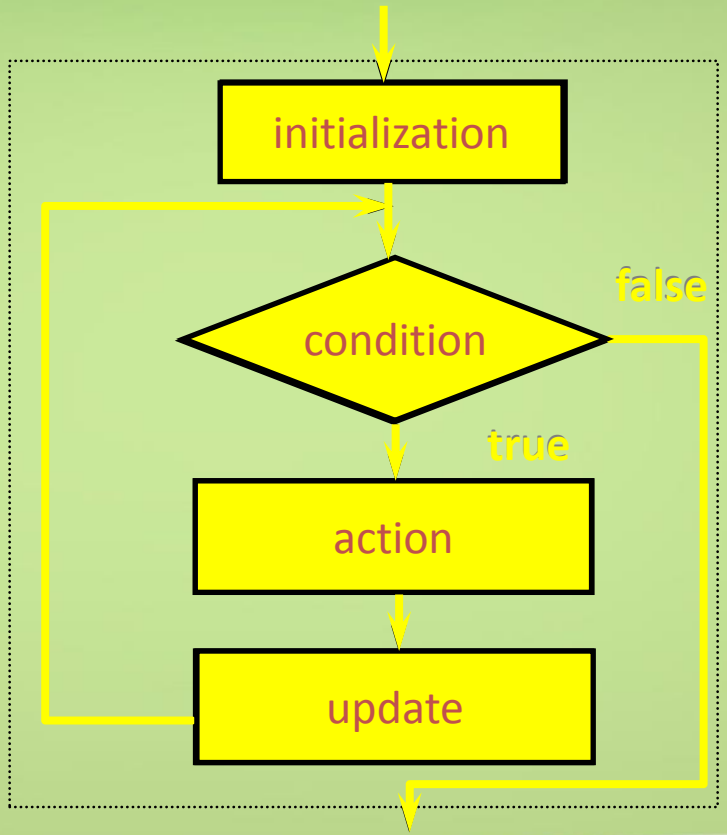
```
for (initialization;
     condition;
     update )
{
     action ;
}
```

- **How does it work?**
  - initialization
  - while condition is true, execute action *and* update



For Loop

```
for (initialization;
     condition;
     update )
{
    action ;
}
```



Compute n! (for loop)

Implement n! using a for loop

- Initialize the intermediate result **t** to 1 (or 0!) and **counter** is initialized in the for loop
- As long as **counter** is less than or equal to n, repeat the computation of **t*counter** and update **counter** using **counter++**

```
public static int factorial(int number) {
  int t = 1; int counter;

  //set up counter, condition check, update together
  for(counter=1; counter<=number; counter++) {
    t *= counter;
  }
  return t;
}
```

Compute n! (for loop)

Implement n! using a for loop

- Initialize the intermediate result **t** to 1 (or 0!) and **counter** is initialized in the for loop
- As long as **counter** is less than or equal to n, repeat the computation of **t*counter** and update **counter** using **counter++**

```
public static int factorial(int number) {
  int t = 1;

  //set up counter, condition check, update together
  for(int counter=1; counter<=number; counter++) {
    t *= counter;
  }
  return t;
}
```

Compute 2^n (for loop)

Implement 2^n using a for loop

- Initialize the intermediate result **t** to 1 (or 2^0) and **counter** is initialized in the for loop.
- As long as **counter** is less than or equal to n, repeat the computation of **t***2 and update **counter** using **counter++**

```
public static int powerTwo(int number) {
  int t = 1;

  //set up counter, condition check, update together
  for(int counter=1; counter<=number; counter++) {
    t *= 2;
  }
  return t;
}
```

Common Mistakes: Floating-point numbers

- Neither use **== (equal)** nor **!= (not equal)** on floating point numbers
- Reasons
  - Floating-point values are approximated
  - In this example, item !=0 may never be false
  - Infinite loop is resulted if the loop condition is always true

```
double item = 1;
double sum = 0;

while (item != 0) {
  sum = sum + item;
  item = item – 0.1;
}
```

Item starts with 1 and is reduced by 0.1 every time the loop body is executed

Common Loop Errors

```
while(balance != 0.0);
{
    balance = balance - amount;
}
// This will lead to an infinite loop!

for(int n=1; n<=count; n++);
{
    IO.outputln("hello");
}
// "hello" only printed once!
```

Which loop to use?

- Programmers are free to choose one of the three loops
- In general
  - **while loop**
    - The number of iterations is unknown (or unclear), and the loop body may not need to be executed
  - **do-while loop**
    - The number of iterations is unknown (or unclear), and the loop body is always executed at least once
  - **for loop**
    - The number of iterations is known (e.g. 100 times)

Array Applications

- Given a list of test scores, determine the average, maximum and minimum scores.
- Read in a list of student names and rearrange them in alphabetical order (sorting).
- Track the ups and downs of a stock index.
- Represent and analysis a digital image as a 2D array.

Array Declaration

- An array is a collection of homogenous data objects.
- Syntax
  - **DataType**[ ] *nameOfVariable*;
    - DataType: The data type of array elements
    - [ ]: We need to write a pair of square brackets next to the data type when declaring an array

Creating an Array

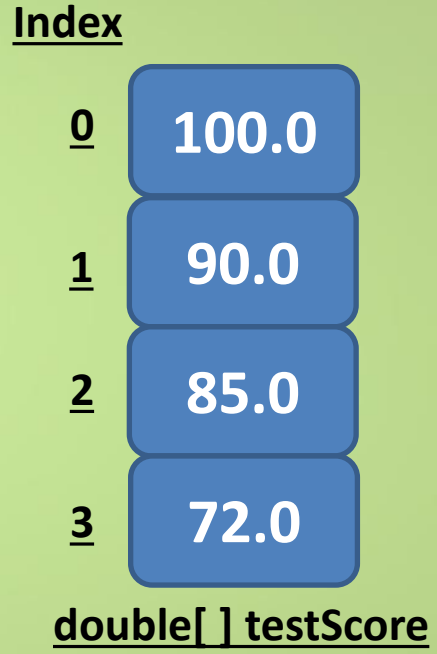
- Some examples of array declarations:
  - double[ ] testScores;
  - int[ ] studentID;
  - double[ ] stockIndex;
- The above declarations do not actually create an array. They declare a reference variable to an array.
- To create an array:
  - testScores = new double[100];
  - studentID = new int[50];
  - double[ ] stockIndex = new double[365]; // create an array during declaration

Initializing an Array

- Declare, define and initialize an array using a single statement:
  - double[ ] testScore = {100.0, 90.0, 85.0, 72.0 };
- This shorthand initialization **must be in ONE statement**
  - For example, the following is wrong:

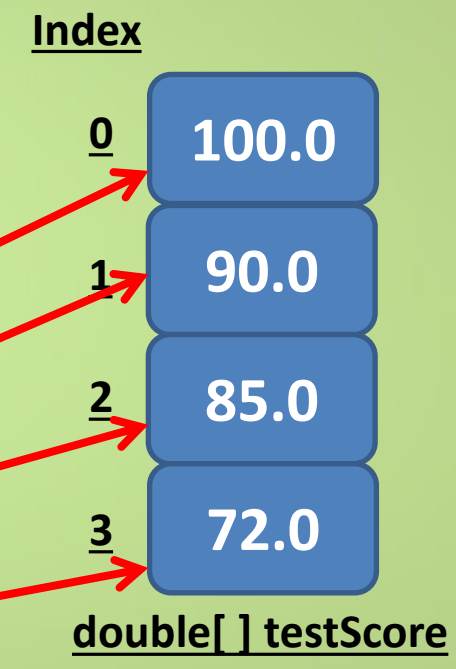
    double[ ] testScore;

    testScore = {100.0, 90.0, 85.0, 72.0};

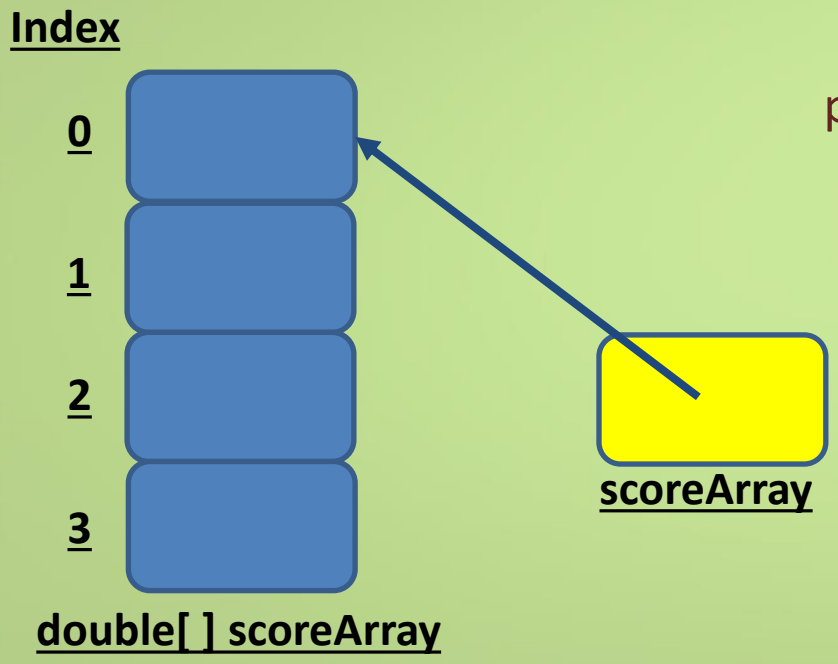


Access an array element

- The array elements are accessed through indices
- The first index starts with 0
  - In the testScore example, we have 4 elements in an array
  - The valid indices are 0, 1, 2 and 3
    - testScore[0]
    - testScore[1]
    - testScore[2]
    - testScore[3]

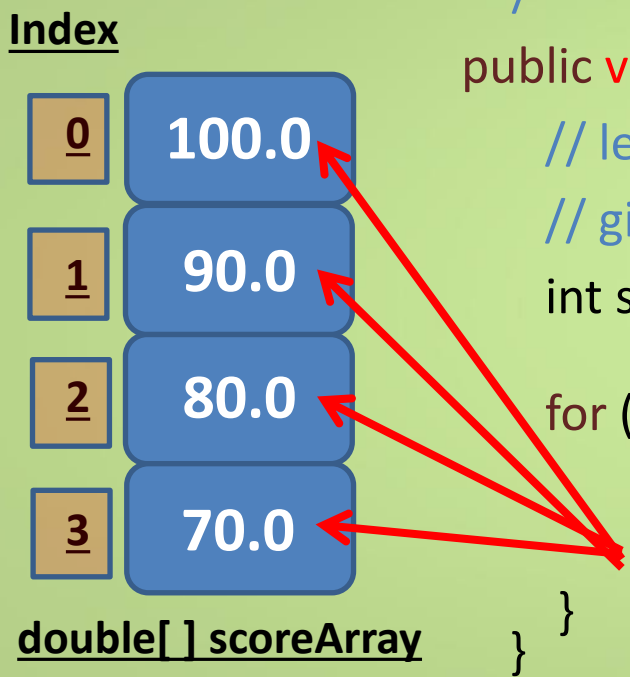


Example



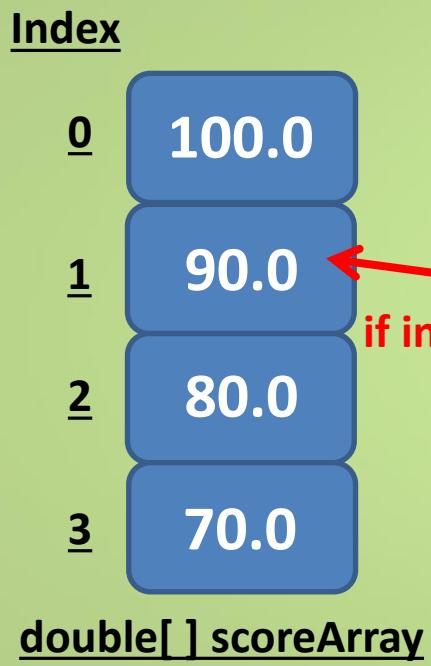

```
/**
 * Create an array for storing a set of scores
 * /
public class Scores {
  double[ ] scoreArray; // declare a reference
                        // variable to an array
  public Scores (int size) {
      scoreArray = new double[size];
  }
```

Example: setScore




```
/*
 * SetScore asks user to enter score for each array element
 */
public void setScore ( ) {
    // length is a constant instance variable for each array that
    // gives the number of elements in the array.
    int size = scoreArray.length;

    for ( int i = 0; i < scoreArray.length; i++ ) {
        IO.output("Enter score for student " + i + ": ");
        scoreArray[i] = IO.inputDouble( );
    }
}
```

Example: getScore




```
/*
  * getScore retrieves the value of an element of the array
  */
public double getScore( int index ) {
    // check to make sure that index is within 0 and array size -1
    if ( index >= 0 && index < scoreArray.length )
        return scoreArray[index];
    else {
        IO.outputln("Error: index out of range");
        return -1;
    }
}
```

if index = 1

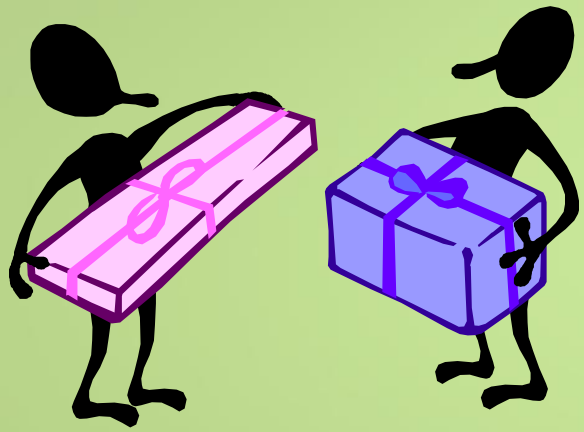
Example: Compute Average

```
/*
 * aveScore computes the average of the values in an array
 */
public double aveScore( ) {
    double sum = 0;      // for storing the cumulative sum
    int size = scoreArray.length;  // size of the array

    for (int i = 0; i < size; i++)
        sum = sum + scoreArray[i];

    return sum / size;
  }
```

Example: Find Maximum

```
/*
  * maxIndex finds the location of the largest values in an array
  * up to index size - 1
 */
 public int maxIndex(int size){
     int mIndex = 0;   // index for the current maximum
     if (size > scoreArray.length) size = scoreArray.length;

     for (int i = 0; i < size; i++) {
        if (scoreArray[i] > scoreArray[mIndex]) mIndex = i;
      }
     return mIndex;
   }
```

Swapping



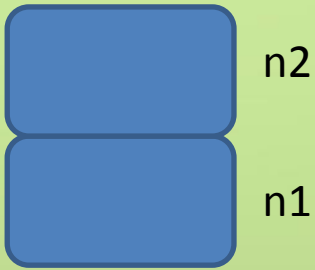
Index	
0	100.0
1	90.0
2	80.0
3	70.0

double[ ] scoreArray

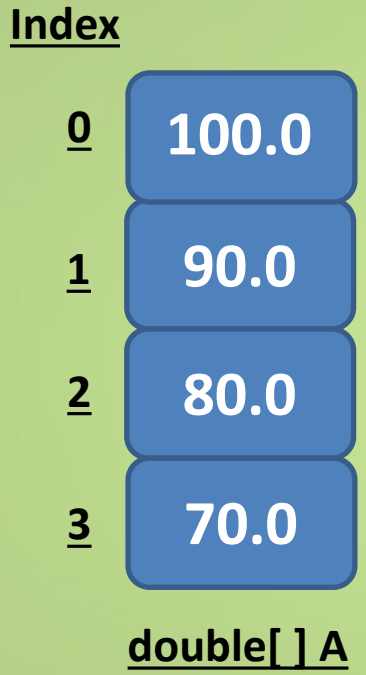
Swapping

```
public void badSwap (double n1, double n2) {
    n1 = n2;
    n2 = n1;
}
```

```
badSwap ( A[1] , A[2] );
```



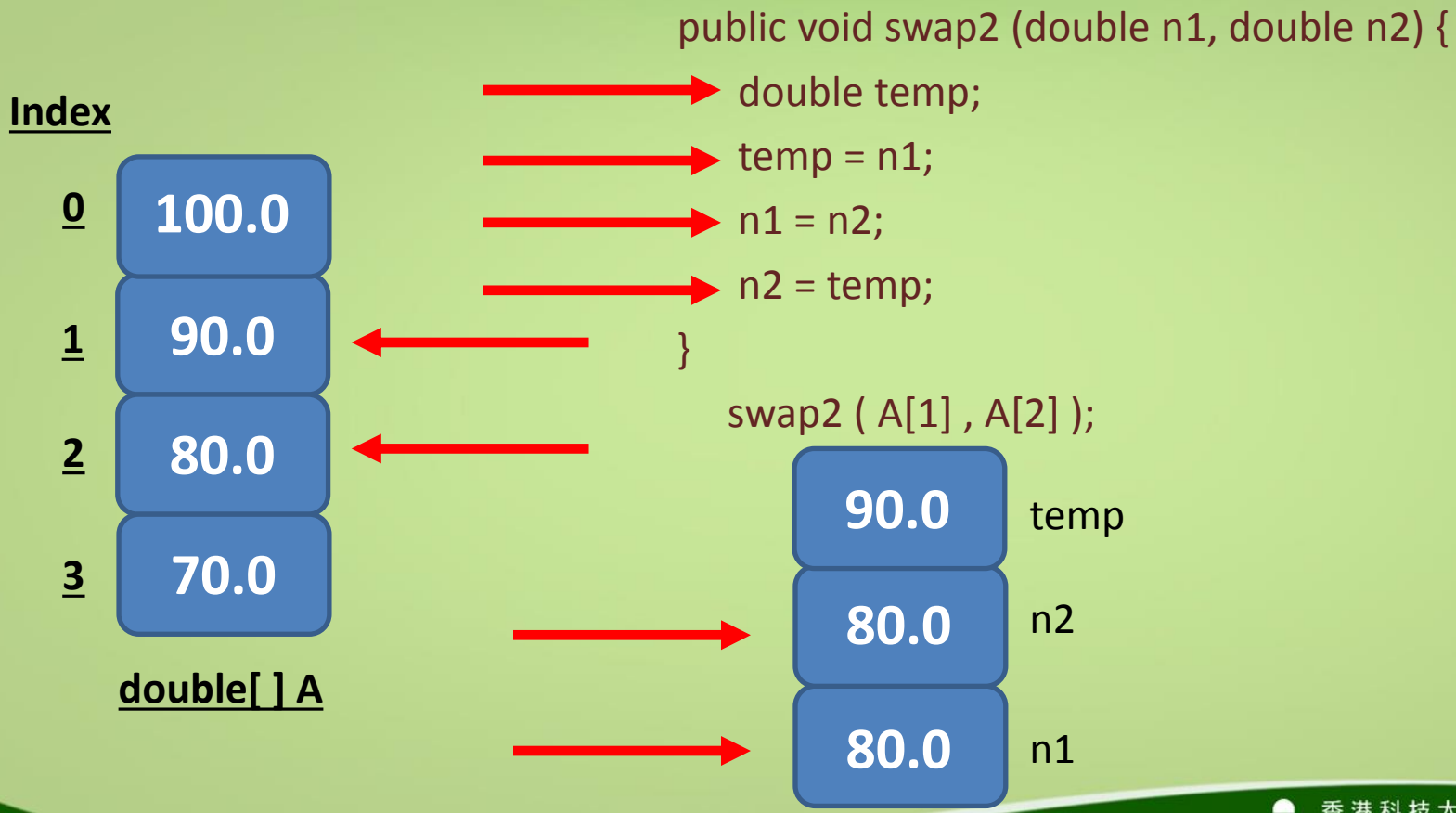
Swapping




```
public void badSwap (double n1, double n2) {
    n1 = n2;
    n2 = n1;
}
```

```
badSwap ( A[1] , A[2] );
```

Swapping



Swapping


```
public void swap (double[ ] arr, int index1, int index2) {
    double temp;                    // double temp;
    temp = arr[index1];             // temp = n1;
    arr[index1] = arr[index2];      // n1 = n2;
    arr[index2] = temp;             // n2 = temp;
}
```

swap (A, 1, 2);