Methods, Arrays and Algorithms

CSC02A2

Outline

Outline

Methods

Method Signature

```
public static int max (int a, int b)
{
      return (a > b) ? a : b;
}
```

Legend:

- modifiers - modify visibility and scope.
- return type - the final value returned after method invocation.
- name - descriptive name of function.
- parameters - number and types of parameters.
- body - code executed when method is invoked[12].

Method Invocation

Using the method from the previous slide we can do the following:

```java
int intMax = max(5, 3);
```

- Value-returning method invocations may be used anywhere a variable of that type could have been used.
- The return value may be ignored.
- Methods with the void return type do not return a value.
- Parameter types as well as order matter.
- Primitive data types are passed by value
- Reference types are passed by reference.

Method Overloading

When multiple methods share the same name but differ in the number, order or type of their parameters then method overloading comes into play. Note that the return type is not included when determining method overloading. Ambiguous invocation is a compile-time error resulting from an inability to distinguish between overloaded methods.

```java
public static int max(int a, int b)
{
  return (a > b) ? a : b;
}
//
public static double max(double a, double b)
{
  return (a > b) ? a : b;
}
// Which max method is called?
double d = max (5, 3);
```

The scope of a variable

- A variable's scope determines where a variable may be legally referenced.
- A variable declared within a method is referred to as a local variable.
- A variable's scope persists from the point it is declared until the end of the block in which it was declared. The use of the same identifier within nested blocks should be avoided.

```java
public static void procrastinate(int t)
{
  long lngTimeWasted = t * 1000000000000000L;
  for(long i = 0; i < lngTimeWasted; i++)
  {
    System.out.println(i);
  }
  System.out.println(i); //Out of scope
}
```

Program Modularity

The use of methods in a program has several advantages:

1. The logic for the method is isolated from the rest of program, this improves program readability and maintainability.
2. Errors relating to the method are largely confined to that method which aids in program debugging.
3. The method now exists as a reusable unit.

Method abstraction: Separation between the use of a method and its implementation (information hiding/encapsulation).

When writing larger programs it necessary to employ step-wise refinement (divide-and-conquer) techniques to software development.

- Top down approach: Focus on high level structure, use stubs for unimplemented methods.
- Bottom up: Test-driven development.

Arrays

Array declaration

Array: fixed-size, sequential collection of elements sharing a common type. In Java as in C++ arrays are 0-indexed.

```java
Type[] refVariable;  //  Or Type refVariable[];

Type[] refVariable = new Type[size];

// Indexing
a[index] = value;
```

`refVariable` only contains a reference to the array in memory. The **`new`** operator results in the actual allocation of the array. Since Java is garbage collected there is no need to manually free the array.

Attempting to index beyond the end of an array or attempting to use a negative index value will result in an `ArrayIndexOutOfBoundsException` being thrown.

Variable length parameter lists

It is possible to create methods which accept an arbitrary length list of parameters of the same type.

```java
public static double add(double... aryNumbers)
{
  double dblTotal = 0.0;
  if(aryNumbers.length == 0)
  {
    System.err.println("Add - no arguments passed");
  }
  for(double d : aryNumbers)
  {
    dblTotal += d;
  }
  return dblTotal;
}
```

The java.util.Arrays class and other useful methods

```java
import java.util.Arrays;

double[] numbers = {2.0, 5.0, 2.1, 9.7, 8.3, 10.0, 12.456, 7.892};
Arrays.sort(numbers, 0, numbers.length); // Sort array
int index = Arrays.binarySearch(numbers, 10.0); // Find index of specific
↪  number
Arrays.fill(numbers, 0); // Fill array with 0

double[] numbersClone = (double[])data.clone(); // Clone full array
System.arrayCopy(numbers, 2, numbersClone, 5, 4); // Partial copy from one
↪  array to another
// This can be used to insert and delete elements at specific indices
```

Sorting Algorithms

Sorting

Definition

The process of arranging an array's elements in ascending or descending order.

Sorting algorithms are classified on the following criteria:

- Computational Complexity (comparisons, swaps, overall)
- Memory usage (in-place or not)
- Approach: Recursive versus Iterative
- Stability: Order of equal entries before sort preserved?
- Method employed (insertion, selection, exchange, merger)

Insertion Sort

Array is divided into two parts: *sorted* and *unsorted*. Sorted section starts with first element (index 0). Each successive entry is then taken from the *unsorted* part and placed in the correct location in the *sorted* part.

- Computational Complexity: Worst - $O(n^2)$, Best - $O(n)$
- Memory usage: In-place
- Approach: Iterative
- Stability: Stable
- Method employed: Insertion

Selection Sort

Select the smallest entry from the list and place it in the first index. Select the next smallest entry and place it at the second index. Repeat the process for the remaining entries.

- Computational Complexity: Worst - $O(n^2)$, Best - $O(n^2)$
- Memory usage: In-place
- Approach: Iterative
- Stability: Stable
- Method employed: exchange

Quick Sort

Quick sort uses a recursive divide and conquer strategy to sort a list. An arbitrary pivot value is selected. The list is then reordered based on the pivot value, all values less than the pivot are placed before the pivot and all values greater than the pivot are placed after the pivot. Recursively sort the less-than values and greater-than values.

- Computational Complexity: Worst - $O(n^2)$, Best - $O(nlogn)$
- Memory usage: $O(n^2)$ but can be in-place as well
- Approach: Recursive
- Stability: Unstable
- Method employed: exchange