

**Passport Program**  
**University of Maryland**  
**College Park**  
**Beginner's Level**

# Overview

- How to Design Computer Programs
- Designing a Problem Solution
- Pseudocode Example
- How Good Is Your Pseudocode
- Pseudocode Elements
- Pseudocode Example
- Suggestions for Implementation
- General Suggestions for Java Programs

# How to Design Computer Programs

- To design programs we want to follow an object-oriented approach
- Object-oriented approach – The solution to a particular program is a collection of objects interacting with one another.
- When designing big systems (e.g., air traffic control system, games, etc.) we need to think about:
  - Classes we need to implement the system
  - Which method each class must have
  - How classes interact with one another
- For now we will not think about which classes we need. We will focus on how to design solutions to relatively small problems. These solutions are usually represented by methods.
- How do we design a solution to a problem. For example, what if someone ask us to write a program that finds sum of a set of integers?

# Designing a Problem Solution

- How do we design a solution to a problem. For example, what if someone ask us to write a program that finds sum of a set of integers?
- Several alternatives exist to come up with a solution to a problem (which we can then turned into a computer program)
  1. **Flowchart:** Graphical representation of the solution, where symbols represent input/output operations, conditionals, and iteration statements.
  2. **Pseudocode:** English-like description of the set of steps required to solve a problem. **This is the alternative we want to use.** It represents our design!

# Pseudocode Example

Pseudocode for finding the minimum value in a set of values

1. Read number of values to process (call this value n)
2. Repeat the following steps until the n input values has been processed
  - a. Read next value into x
  - b. If (x is the first value read)  
    currentMinimum = x  
    else {  
        if (x < currentMinimum)  
            currentMinimum = x  
    }
3. Print currentMinimum value

# How Good Is Your Pseudocode

You can determine if your pseudocode is correct by using the following guidelines:

- Your code does not use language constructs that are particular to a programming language (e.g., JOptionPane).
- If you provide the pseudocode to a person the person will not need to ask you questions in order to transform the pseudocode into code.

As much as possible try to create a trace table to test your pseudocode.

# Pseudocode Elements

- When writing pseudocode you need three fundamentals constructs:  
Input, output, assignments, repetition structures, conditionals
- To help you with the design of pseudocode you can use the following syntax to represent the above constructs:

■ Input	variable = read()	e.g., x = read()
■ Output	print(variable)	e.g., print(x)
■ Assignment,	x = <value>	e.g., x = 20, s = "Bob"
■ Repetition		
	while (expression) {	OR
	statements	
	}	do {
		while (expression)
■ Conditional		
	if (expression) {	OR
	statements	
	}	if (expression1) {
		statements
		} else if (expression2) {
		statements
		} else {
		statements
		}

- For comparisons use: ==, <, >, <=, >=, !=
- Notice the above constructs look like Java code but they are not java code. Use them to define your pseudocode.

# Pseudocode Example

- Pseudocode

```
entries = read()
prev = read()
i = 1
valid = true
while (i < entries && valid) {
    curr = read()
    if (curr < prev) {
        valid = false;
    }
    prev = curr
    i = i + 1
}
print("Sequence is:")
Print(valid)
```

- Run always a trace table to verify your pseudocode is correct
- Could you implement the above code without actually knowing what it does??

# Suggestions for Implementation

- **Make sure you have written pseudocode**
- **Do not wait until the last minute** – Code implementation could be unpredictable.
- **Incremental code development** – Fundamental principle in computer programming. Write a little bit of code, and make sure it works before you move forward.
- **Don't make assumptions** – If you are not clear about a language construct write a little program to familiarize yourself with the construct
- **Good Indentation** – From the get-go use good indentation as it will allow you to understand your code better

# Suggestions for Implementation

- **Good variable names** – Use good variable names from the get-to.
- **Testing** – Test your code with simple cases first.
- **Keep backups** – As you make significant progress in your development, make the appropriate backups.
- **Use a debugger** – Later on we will see this tool that allow us to find bugs (problems) in your program. For now, you can use `System.out.println` for debugging. It will allow you to
  - Determine values of variables
  - Determine the execution path
- **Take breaks** – If you cannot find a bug take a break and come back later

# General Suggestions for Java Programs

- Name your variables using lowercase letters. If a variable has two words (e.g., waterTemperature) use a lowercase character to indicate the beginning of the second word. The following are valid variable names:

oilPressure, firstSystemIteration, row

- Class names must start with capitals. The following are valid class names:

House, Car, ElectricCar

- Constants use uppercase letters and \_  
HOURS\_PER\_DAY, DAYS\_PER\_WEEK

# General Suggestions for Java Programs

- Define the general framework of your class and main method
- When writing loops set the loop skeleton (making sure the loop ends) before you add additional statements to the body of the loop.
- When writing expressions for if statement define the skeleton first

```
if ( () || () || () ) {  
    }  
}
```

Now fill the () with the required expressions.

If the expressions are complex you can follow this approach:

```
if ( ( ) ||  
      ( ) ||  
      ( ) )
```