

Top-Down Design

Course: CPSC 1150
Instructor: Dr. Bitá Shadgar

Lecture 14

Learning Outcomes

- Define method abstraction and name its benefit
- Define stepwise refinement or modularization
- Apply top-down design
- Evaluate a design
- Implement a design using top-down or bottom-up implementation

Method abstraction

Definition

Abstraction is the process of considering something independently of its associations, attributes, or concrete accompaniments.

- Method abstraction means considering methods without the details of their implementation
- A method can be thought of as a black box
 - May accept some inputs
 - May return a value
 - Statements inside are hidden
- Method abstraction is a very useful design tool

Stepwise refinement

Definition

Stepwise refinement is a technique for developing a large program by decomposing problems into sub-problems.

- Applies method abstraction to the software development process
- Also known as "divide and conquer" or "modularization"
- Called 'stepwise' because one **level of decomposition** is usually not enough
 - The sub-problems should be further decomposed until each bottom-level problem is very small and manageable

Example

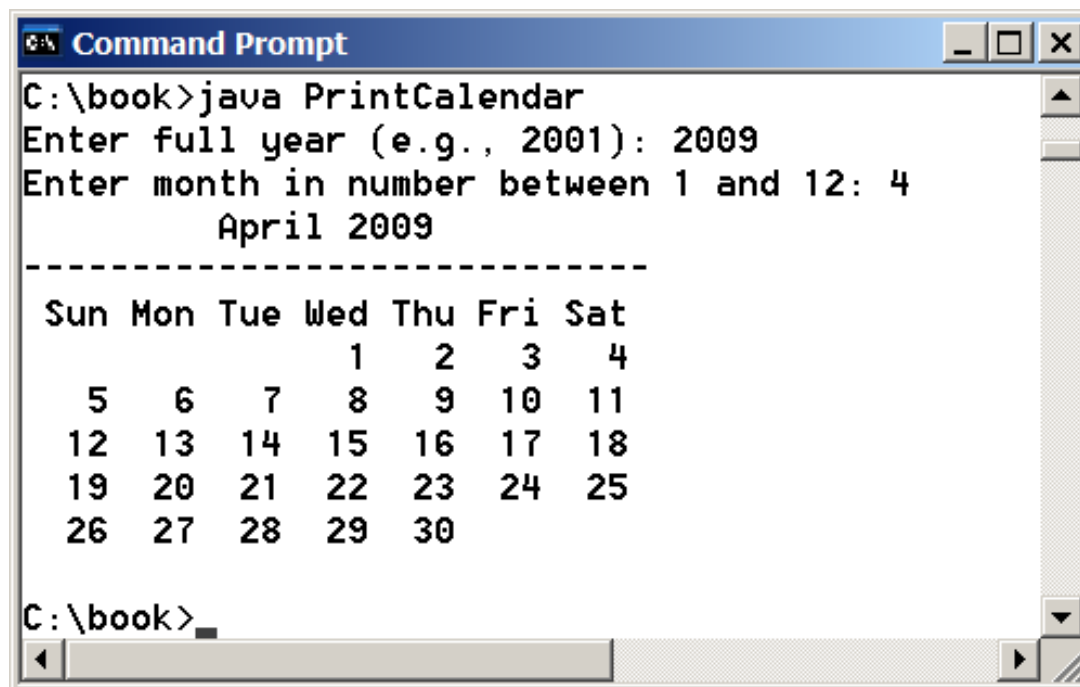
Let's check out the program PrintCalendar from the textbook and think about how to divide it into sub-problems.

Top-down design diagram

- A top-down design diagram is a tool to help illustrate the results of performing stepwise refinement
 - Also helpful in understanding the structure of a modular program
- Big-picture flowchart - doesn't consider details of implementation
- How to draw:
 - General problem in a box at the top
 - Each sub-problem in its own box, one level lower
 - Connect problems to their sub-problems
- Sub-problems usually correspond to methods
 - Some sub-problems are too simple to require their own methods

Case Study - PrintCalendar

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.

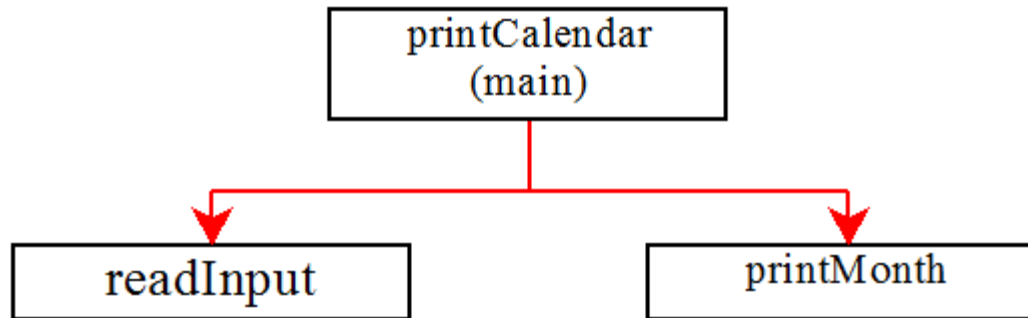


```
Command Prompt
C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
      April 2009
-----
Sun Mon Tue Wed Thu Fri Sat
    1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30

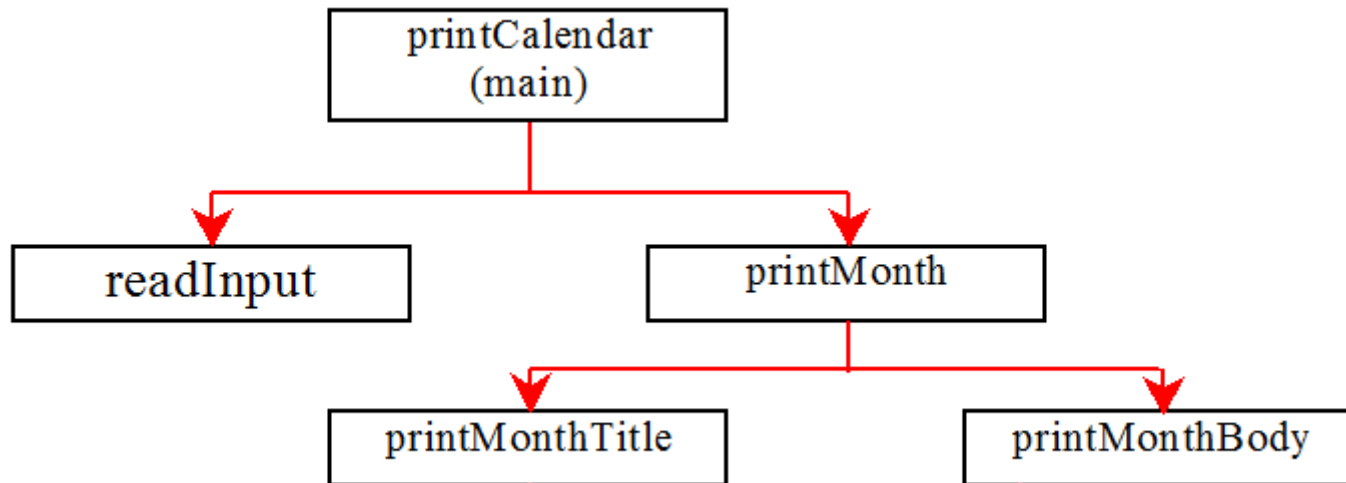
C:\book>
```

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The user has entered the command `java PrintCalendar` at the `C:\book>` prompt. The program has prompted for a full year (2009) and a month (4), and has displayed the calendar for April 2009. The calendar is formatted with a header row for the days of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat) and a grid of dates. The dates are arranged in rows, with the first row starting on Wednesday (1st) and ending on Saturday (4th). The second row starts on Sunday (5th) and ends on Saturday (11th). The third row starts on Sunday (12th) and ends on Saturday (18th). The fourth row starts on Sunday (19th) and ends on Saturday (25th). The fifth row starts on Sunday (26th) and ends on Saturday (30th). The prompt `C:\book>` is visible at the bottom of the window.

Design Diagram



Design Diagram

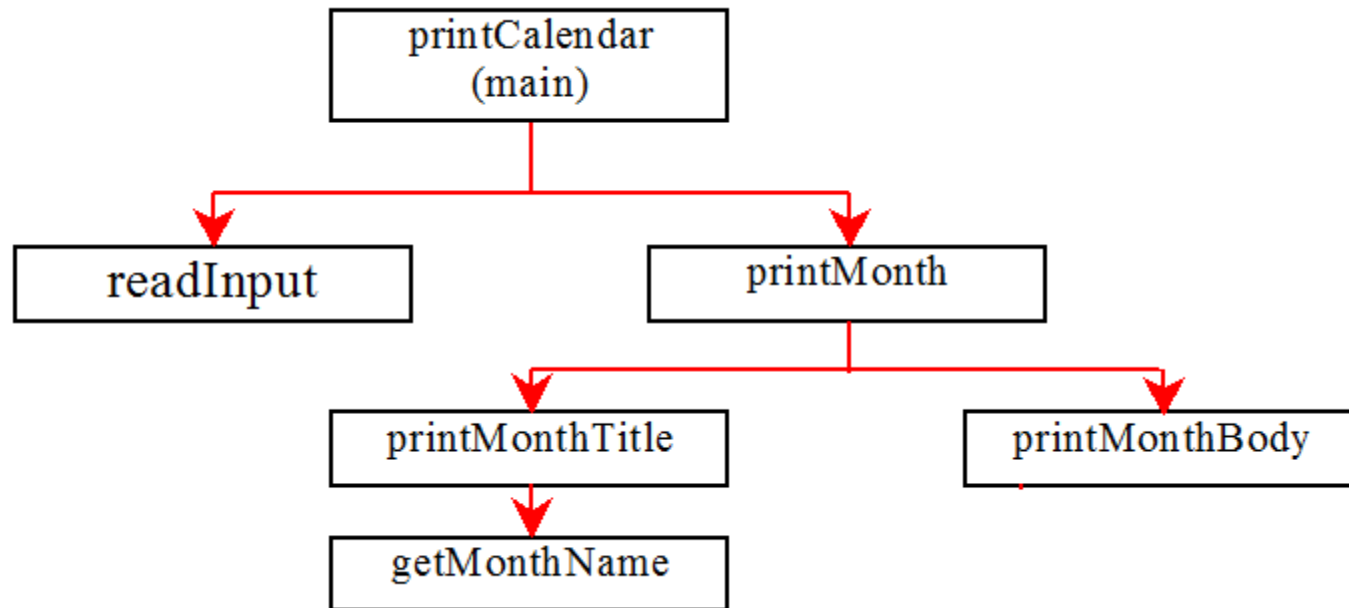


Command Prompt

```
C:\book>java PrintCalendar
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 4
  April 2009
-----
Sun Mon Tue Wed Thu Fri Sat
    1  2  3  4
  5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

The screenshot shows the program's output. The title "April 2009" is highlighted with a red box and labeled "Title". The calendar grid is highlighted with a blue box and labeled "Body".

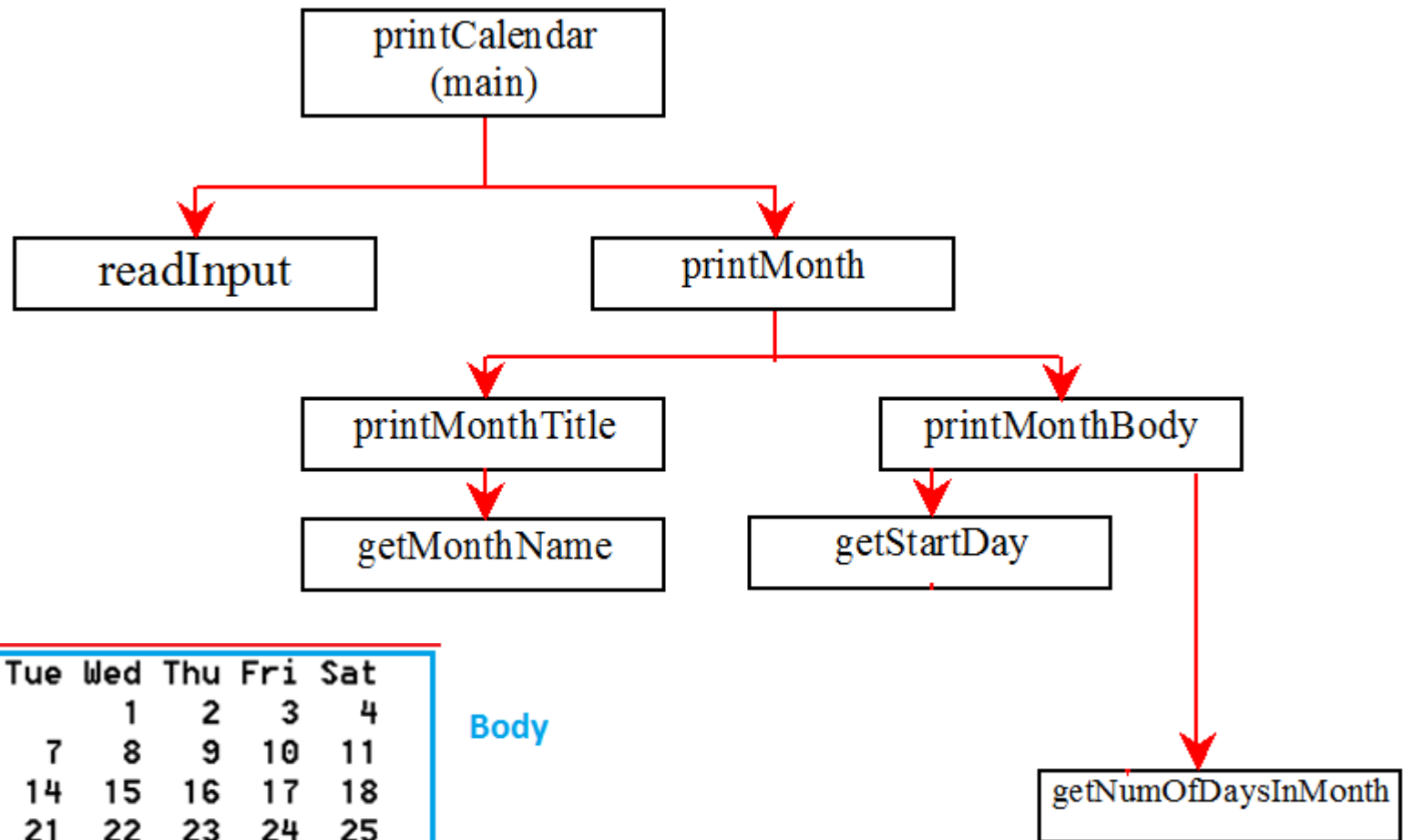
Design Diagram



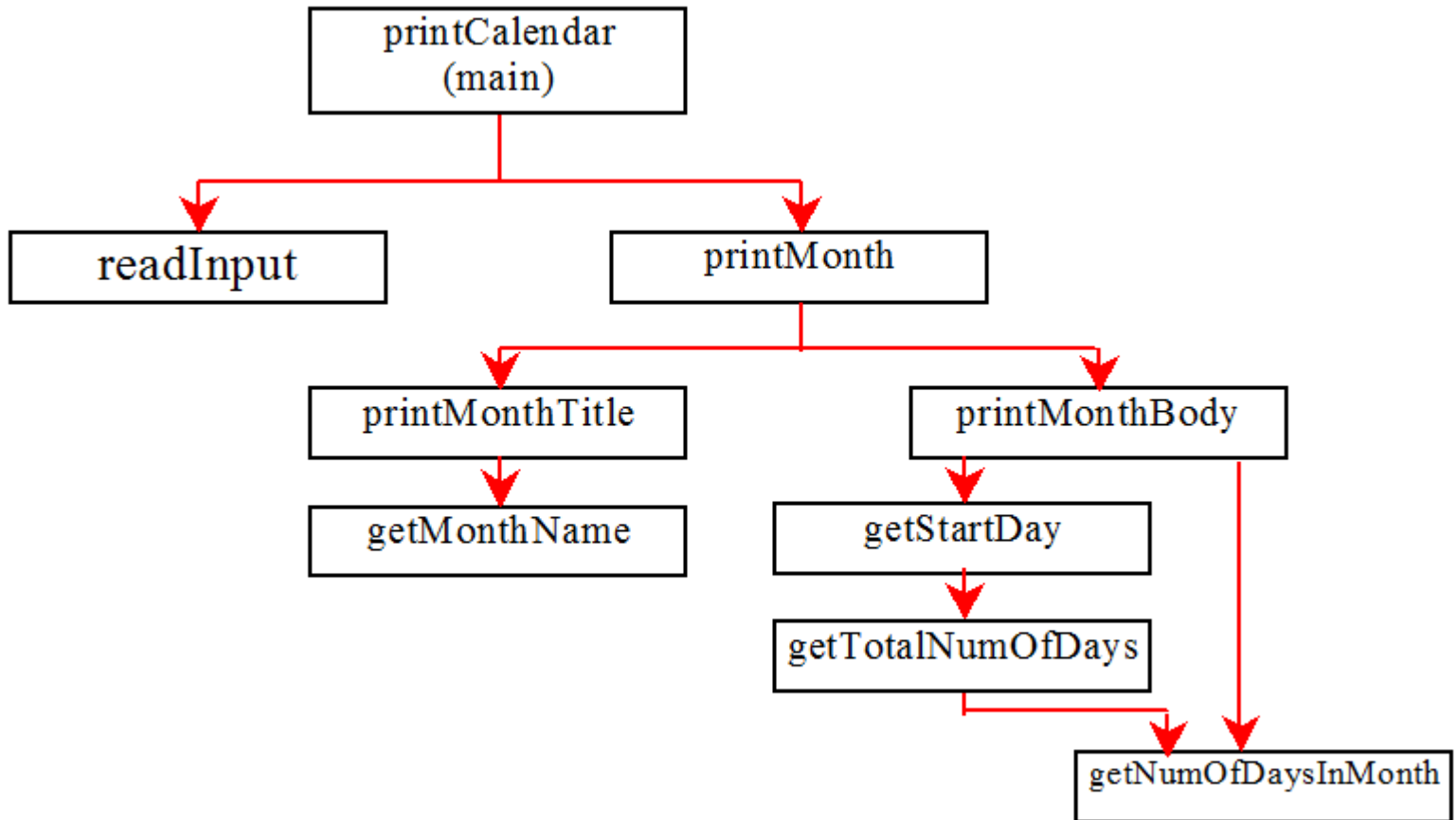
printMonthTitle
April 2009

Title

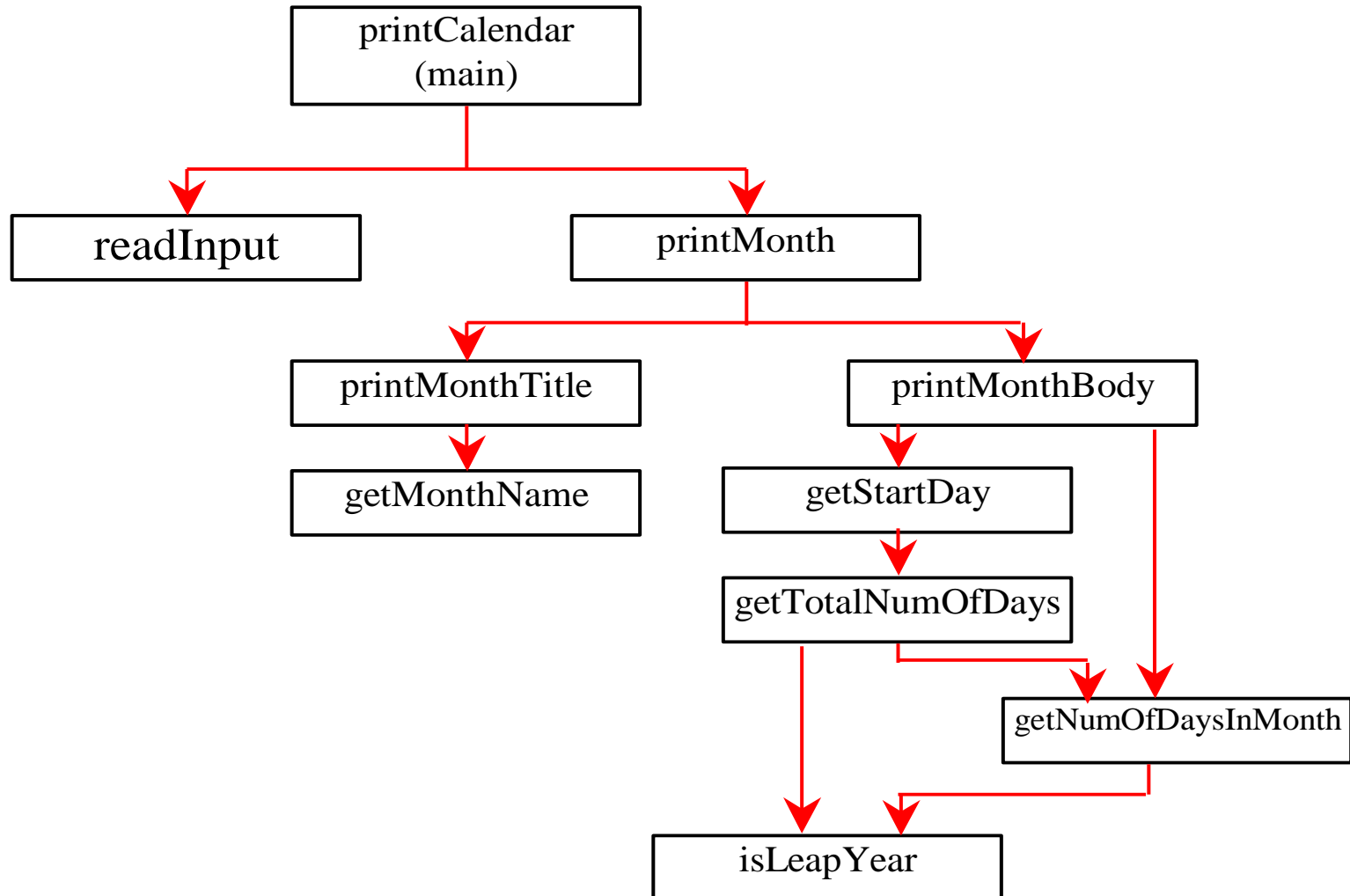
Design Diagram



Design Diagram



Design Diagram



Design Diagram - Prime Numbers

- Design a program that given a positive integer number, limit, from user, it prints all the prime numbers less than limit, in a tabular output, where each prime number is written in 10 character-space and only 4 prime numbers per line.
- The following shows a sample run:

Enter a positive number as your limit? 110

2	3	5	7
11	13	17	19
23	29	31	37
41	43	47	53
59	61	67	71
73	79	83	89
97	101	103	107
109			

Design Diagram

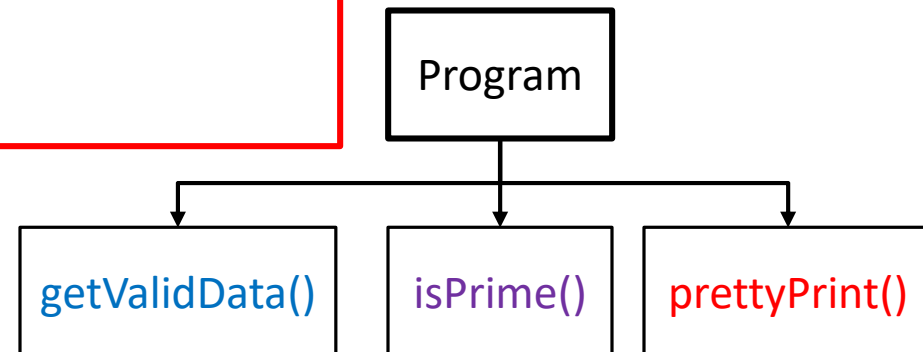
Enter a positive number as your limit? 110

2	3	5	7
11	13	17	19
23	29	31	37
41	43	47	53
59	61	67	71
73	79	83	89
97	101	103	107
109			

isPrime(num)

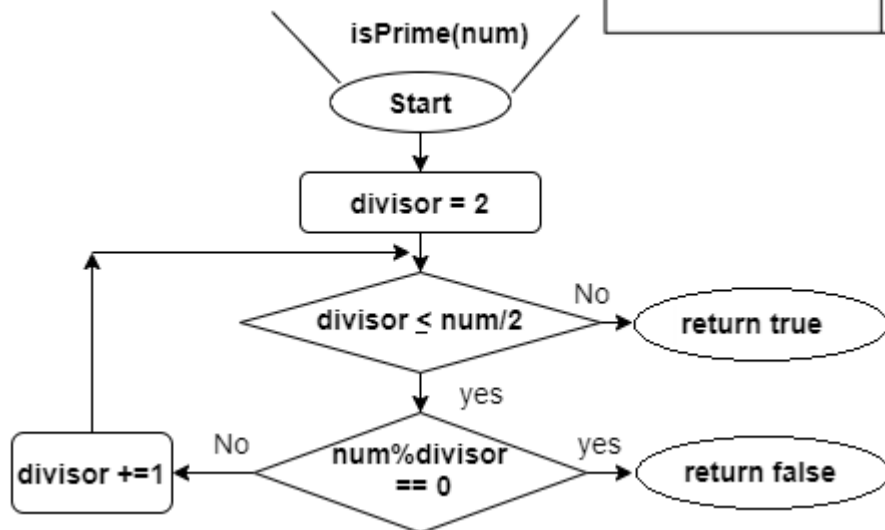
limit = getValidData()

prettyPrint(num, counter)



isPrime function

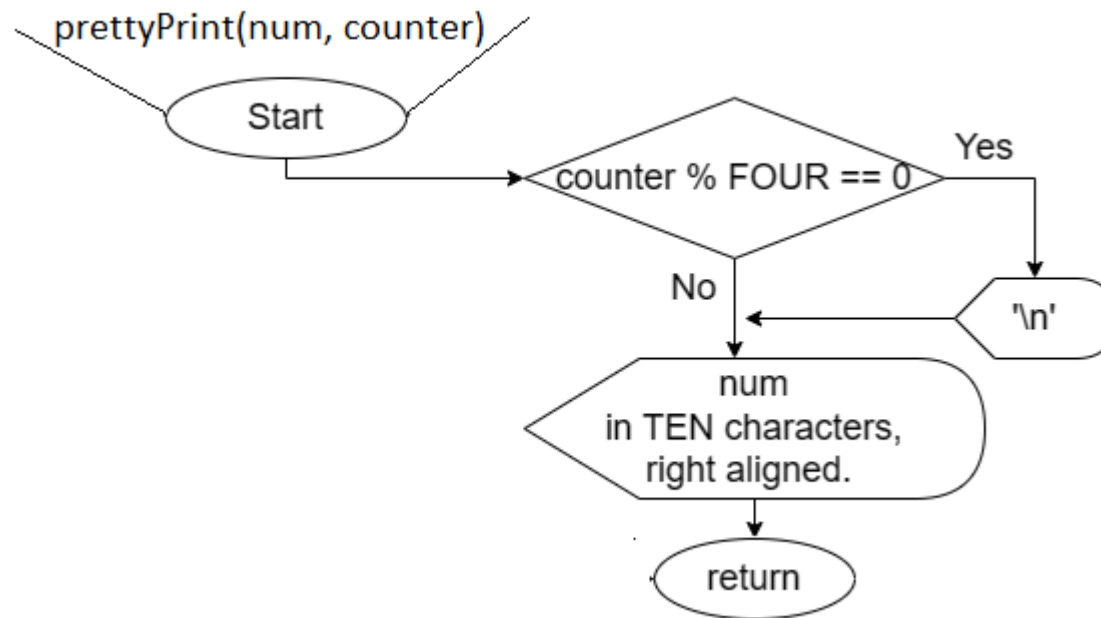
IPO chart for <code>isPrime</code> function		
Input	Processing	Output
An integer number, named num	Check if the input is prime or not	true if number is prime, otherwise false
	Divide num by numbers from 2 up to num/2, if the remainder is zero, return false Otherwise, num is Prime and return true.	



prettyPrint function

IPO chart for **prettyPrint** function

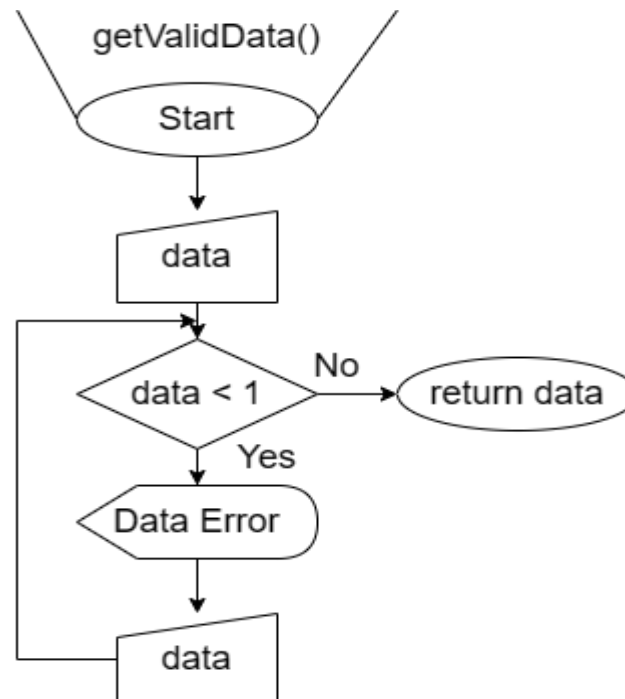
Input	Processing	Output
An integer named num An integer named counter	Prints the input in a tabular format, each number in 10 character-space and only 4 prime numbers per line.	nothing



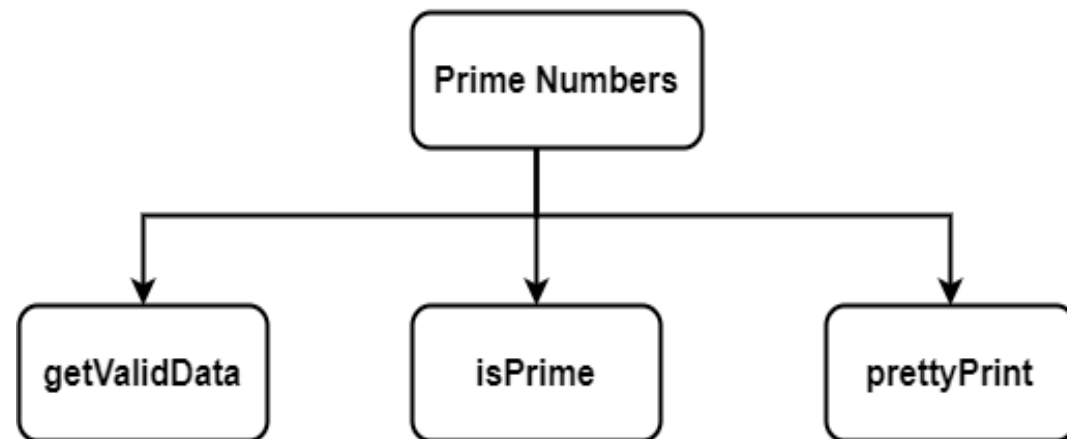
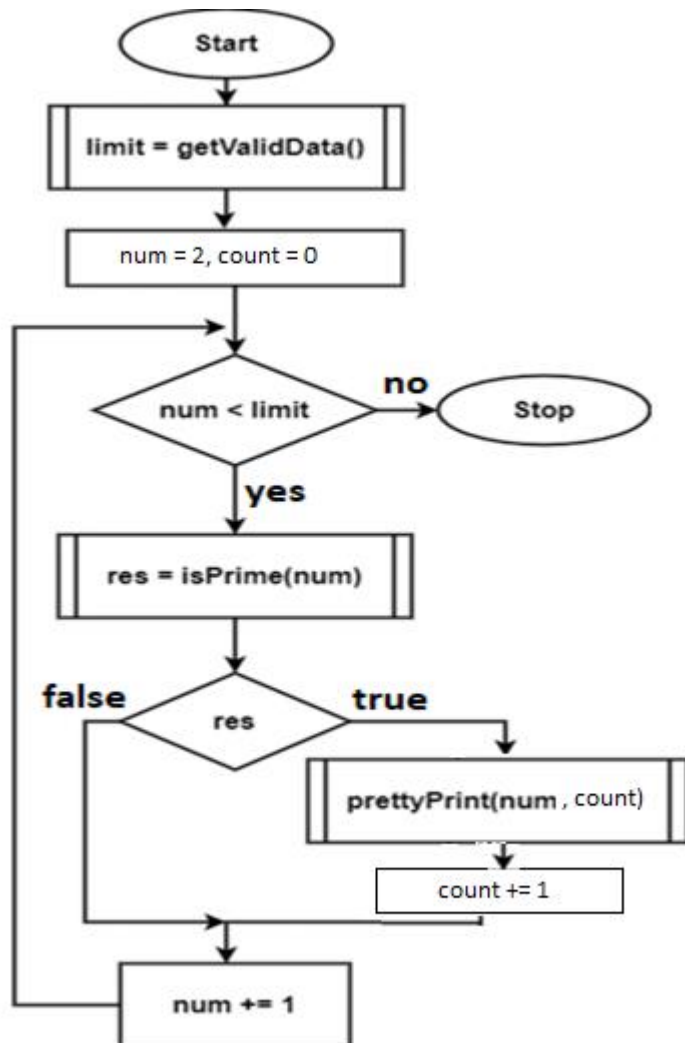
getValidData function

IPO chart for **getValidData** function

Input	Processing	Output
nothing	Asks user for a positive integer number	A valid integer number



Flowchart vs Design Diagram



Evaluating a design

- Good designs have **strong cohesion** and **loose coupling**

Definition

Cohesion is the degree to which each method does exactly one “thing”

- Should be able to describe what a method does in one sentence
- If you have to say `and', you should probably split it up

Definition

Coupling is the degree to which data is passed around

- Want low communication complexity
- Information should only be available on a need-to-know basis
- Aim for fewer parameters and fewer class variables

Implementing a design

- Implementation is a lot easier when you are working from a modular design
- Implement each method **one at a time**
 - Need to test each method before implementing the next one
- Two main approaches:
 - Top-down implementation
 - Bottom-up implementation
- Can use whichever approach you prefer

Top-down implementation

- Implement the methods in the diagram one by one, going from top to bottom

Question: How can you implement an upper-level method without implementing the methods it will call?

Definition

A **stub** is a simple but incomplete version of a lower-level method which can be used to test higher-level methods.

- For each method:
 - Implement the method.
 - Implement a stub for each method invoked inside.
 - Test the method until correct.

Example of Stub

```
import java.util.Scanner;

public class Circle{
    final static double PI = 3.1415;
    public static void main(String[] args){
        double radius, area, perimeter;

        radius = getInput();
        area = calArea(radius);
        perimeter = calPerimeter(radius);
        printResult(radius, area, perimeter);
    }
```

```
//stub for getting input
public static double getInput(){
    //Todo later
    return 4.0;
}

public static double calArea(double r){
    return r*r*PI;
}

public static double calPerimeter(double r){
    return 2*r*PI;
}
} //end of class
```

```
public static void printResult(double r, double a, double p){
    System.out.printf("[radius = %.2f, perimeter = %.2f, area = %.2f]\n", r, p, a);
}
```

More Practice – War Card game

- Design War Card game that in each round the program asks a valid number from user that stand for a card, and also randomly selects a card for computer. Then it declares the cards that user and computer have chosen and the winner. The card with higher value is the winner and gets one score. If they are tie, the user's total score must be doubled. The program must be executed 10 rounds and then declares the total score of user.
- Ace, Jack, Queen and King must be considered as 1, 11, 12 and 13 respectively. The suit doesn't matter.

Sample Output

==> Enter your card : 12

R1: Computer card is 10 of Diamonds, User card is Queen of Hearts

User wins - user score is 1

==> Enter your card : 46

R2: Computer card is Ace of Diamonds, User card is 7 of Clubs

User wins - user score is 2

==> Enter your card : 36

R3: Computer card is 10 of Hearts, User card is 10 of Spades

It's tie - user score is 4

==> Enter your card : 16

R4: Computer card is 9 of Diamonds, User card is 3 of Diamonds

Computer wins - user score is 4

...

==> Enter your card : 18

R10: Computer card is 4 of Spades, User card is 5 of Diamonds

==> User wins - user score is 19

End of Game!