Program Development Cycle

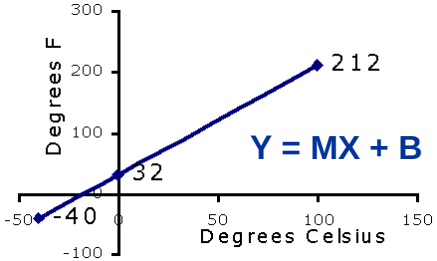
- Design Phase
  - Write Program Specifications
    - Analysis of requirements
    - Program specifications description
      - Describe what the goals of the program
      - Describe appearance of input and output
  - Algorithm Design
    - Mathematical Analysis and Algorithm
    - Flow Chart, Pseudo-code or UML diagram to model program
  - Verify algorithm
    - Test with known data
    - Solve manually
- Implementation Phase
  - Convert Design into Java Code
    - Use Compiler to remove Syntax Errors
  - Use known test data to verify program works
    - Debug and eliminate logic errors

 1

Specifications and Math Algorithm Design

- Program Specifications
  - This program will convert from ºC to ºF
  - Design input and output appearance
- Mathematical Analysis and Algorithm Design
  - Boiling point
    F =212
    C = 100
  - Freezing point
    F = 32
    C = 0

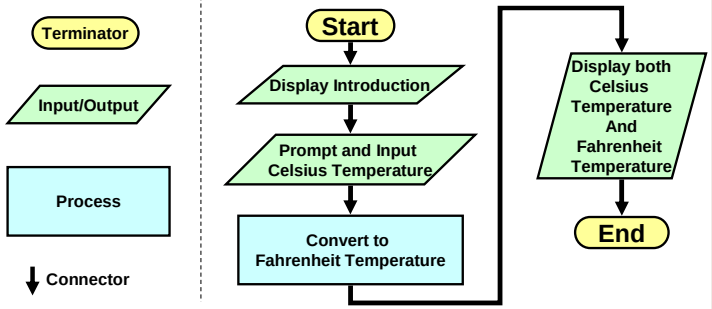
F = (180 / 100) C + 32
= (9/5) C + 32
= 1.8 C + 32



 2

Sequential Algorithm Design

- Flowcharts are an excellent way to plan the sequence of operations that the program will need to perform



- Pseudo-code can also be used for algorithm design, but program flow must be understood

 3

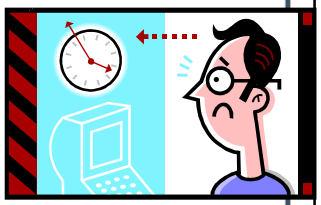
Verify Math and Sequential Algorithms

- Testing with known data
  - Boiling point F =212 C = 100
  - Freezing point F = 32 C = 0
  - Collect Data
    - Internet weather report
    - Radio weather report
- Be the Computer
  - Do the sequential process specified in you design
    - Flow Chart or Pseudo-code
  - Solve mathematics manually using calculator
    - Test Data needs to test every path
    - Minimum two sets of data for every path

 4

Implementation Phase

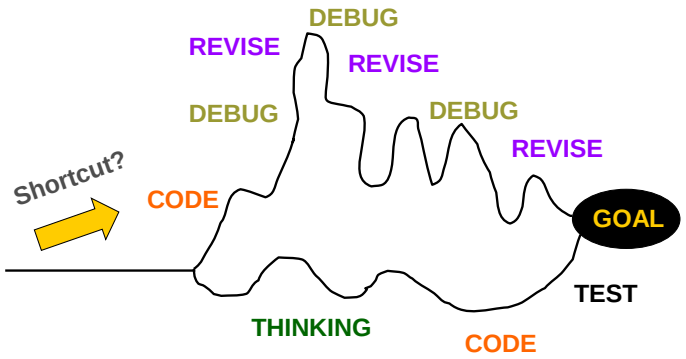
- Translate Algorithm into Java Code
  - Compile to detect and remove *syntax errors*
- Run Program
  - Test with known data
  - Test data must test every control path of program
  - Detects program *logic errors*



- May require redesign
  - Re-evaluation of specifications and algorithms
  - Time spent on a program is dependent on the thoroughness of your design phase
  - Can program be improved?

 5

Coding First Is No Shortcut?



The more time you spend in design phase the less time you will waste debugging code that cannot work

 6

Common Java Programming Errors

- Allow enough time to design the program
  - Test your design before you write code
- Most common Java coding errors
  - Forgetting to save program with same file name as class name used within program
  - Omitting semicolon `;` at end of each statement
  - Forgetting `\n` to indicate new line
  - Forgetting to put the matching closing brace `}`
- Proper programming style prevents errors
  - Tab sections contained in braces for readability
  - Use Next-line style as utilized in these slides
  - Use accepted identifier naming convention
  - Use comments to explain your code

 7

Flow of Control

- Definition: The sequential execution of statements in a program
  - Sequential Control Structure (Top-Bottom)
    - It is characterized by a flow chart construct without branches
  - Selection Control Structure (Branching)
    - Decision making control
    - Tests an Assertion Statement
      - Evaluated as True or False (Humans)
      - Evaluated as Yes or No (Humans)
      - Evaluated as 1 or 0 (Computers)

 8

Relational Operators

- Relational operators are used to compare two data objects.
- The result of the comparison is either true or false.

==	Equal to	!=	Not Equal to
>	Greater	>=	Greater or Equal
<	Less	<=	Less or Equal

- Note the difference between == and = operator

Logical Operators

- Used in while and if assertions true/false
- There are three logical operators
  - AND &&
  - OR ||
  - NOT !

A	B	A && B
F	F	F
F	T	F
T	F	F
T	T	T

A	B	A || B
F	F	F
F	T	T
T	F	T
T	T	T

A	!A
F	T
T	F

Note on Precedence: Evaluate relational first and then logical

Operators Precedence (Highest to Lowest)

( )	Defines order of operation
-	Minus (unary)
(int) (double)…	Type cast operators
!	Logical NOT (unary)
* / %	Multiply, Division, Remainder
+ -	Addition&Concatenation, Subtraction
< <= > >=	Relational Operators
== !=	Relational Operators
&& ||	Logical Operators AND OR
=	Assignment

Relational Logical Operator Examples

- Given:

```
int     nA = 23, nB = 16;
char    cEntry = 'y';
```

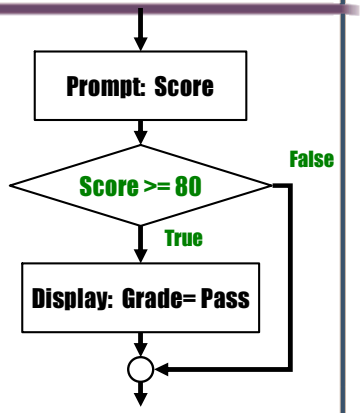
- Then determine if true or false:

```
nA > nB
nA < nB
nA >= nB
nA <= nB
nA != nB
nA == nB
nA - nB < nB
nA >= 0 && nB < nA
```

```
(nA < 5) && (nB > 10)
(cEntry=='y') || (cEntry=='Y')
sUpEntry.equals("YES")
```


if Selection Control Structure

- Characterized by a diamond shaped flow chart symbol, containing an assertions with two possible outcomes branches (true false)

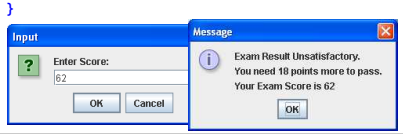


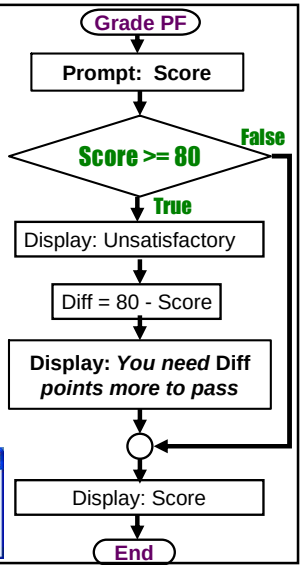

```
if(Score >= 80)
    System.out.println("Grade = Pass");
```

 13

Compound if Selection Control Structure

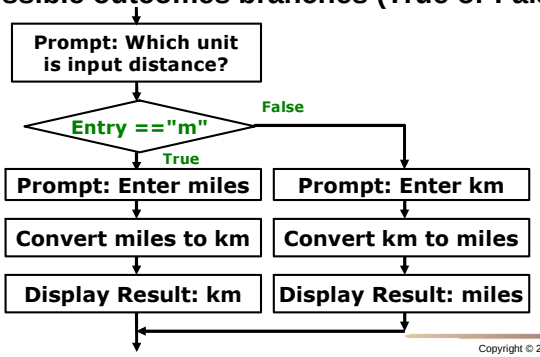
```
1.  import javax.swing.*;
2.  public class GradePF
3.  {
4.    public static void main(String args[])
5.    {
6.      int nScore, nDiff;
7.      String sEntry, sResult = "";
8.      sEntry = JOptionPane.showInputDialog(
9.        "Enter Score:");
10.     nScore = Integer.parseInt(sEntry);
11.     if(nScore < 80)
12.     {
13.       sResult = "Exam Result Unsatisfactory";
14.       nDiff = 80 - nScore;
15.       sResult = sResult + "\nYou need "
16.         + nDiff + " points more to pass.\n";
17.     }
18.     sResult = sResult + "Your Exam Score is "
          + nScore;
19.     JOptionPane.showMessageDialog(null, sResult);
20.   }
21. }
```





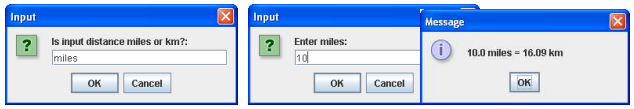
if - else Selection Structure

- Characterized by a diamond shaped flow chart construct, containing an assertions with two possible outcomes branches (True or False).

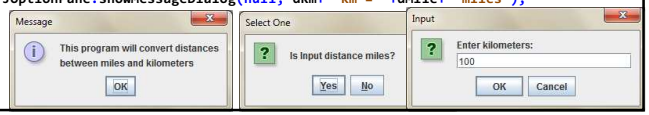


 15

```
1.  import javax.swing.JOptionPane;
2.  public class MileKm
3.  {
4.    public static void main(String args[])
5.    {
6.      double dMile, dKm;
7.      String sEntry, sLowEntry;
8.      char cSelect;
9.      sEntry = JOptionPane.showInputDialog("Is input distance miles or km?:");
10.     sLowEntry = sEntry.toLowerCase();
11.     cSelect = sLowEntry.charAt(0);
12.     if(cSelect == 'm')
13.     {
14.       sEntry = JOptionPane.showInputDialog("Enter miles:");
15.       dMile = Double.parseDouble(sEntry);
16.       dKm = dMile * 1.609;
17.       JOptionPane.showMessageDialog(null, dMile+" miles = "+dKm+" km");
18.     }
19.     else
20.     {
21.       sEntry = JOptionPane.showInputDialog("Enter kilometers:");
22.       dKm = Double.parseDouble(sEntry);
23.       dMile = dKm / 1.609;
24.       JOptionPane.showMessageDialog(null, dKm+" km = "+dMile+" miles");
25.     }
26.   }
27. }
```

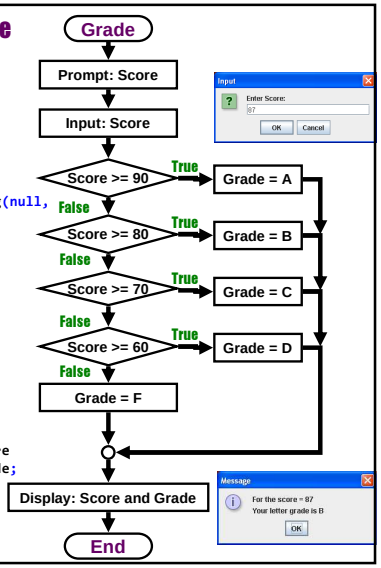


```java
import javax.swing.JOptionPane;
public class MileKm2
{
  public static void main(String args[])
  {
    double dMile, dKm;
    String sEntry, sLowEntry;
    char cSelect;
    int nSelect;
    JOptionPane.showMessageDialog(null, "This program will convert distances\n"
      + "between miles and kilometers");
    nSelect = JOptionPane.showConfirmDialog(null,"Is Input distance miles?",
      "Select One", JOptionPane.YES_NO_OPTION);
    if(nSelect == JOptionPane.YES_OPTION)
    {
      sEntry = JOptionPane.showInputDialog("Enter miles:", "0");
      dMile = Double.parseDouble(sEntry);
      dKm = dMile * 1.609;
      JOptionPane.showMessageDialog(null, dMile+" miles = "+dKm+" km");
    }
    else
    {
      sEntry = JOptionPane.showInputDialog("Enter kilometers:", "0");
      dKm = Double.parseDouble(sEntry);
      dMile = dKm / 1.609;
      JOptionPane.showMessageDialog(null, dKm+" km = "+dMile+" miles");
    }
  }
}
```



if - else if - else Selection Structure

```java
import javax.swing.JOptionPane;
public class Grade
{
  public static void main(String args[])
  {
    int nScore;
    char cGrade;
    String sEntry, sOutput;
    sEntry = JOptionPane.showInputDialog(null,
      "Enter Score:");
    nScore = Integer.parseInt(sEntry);
    if(nScore >= 90)
      cGrade = 'A';
    else if(nScore >= 80)
      cGrade = 'B';
    else if(nScore >= 70)
      cGrade = 'C';
    else if(nScore >= 60)
      cGrade = 'D';
    else
      cGrade = 'F';
    sOutput = "For the score = " + nScore
     + "\nYour letter grade is " + cGrade;
    JOptionPane.showMessageDialog(null,
      sOutput);
  }
}
```



switch case Selection Structure

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. If the break statement is not present, the next case statement will also be executed.

```
switch (switch-expression)
{
 case value1:  statement(s)1;
        break;
 case value2: statement(s)2;
        break;
 …
 case valueN: statement(s)N;
        break;
 default: statement(s)-for-default;
}
```

The default case, is optional, and performs actions when none of the specified cases matches the switch-expression.

 19

switch - case Selection Structure

```java
import javax.swing.JOptionPane;
public class SwitchCaseConfirmDialog
{
  public static void main(String[] args)
  {
    int nOption;
    nOption = JOptionPane.showConfirmDialog(null, "Do you like programming",
      "Make your choice", JOptionPane.YES_NO_CANCEL_OPTION);
    switch (nOption)
    {
      case JOptionPane.YES_OPTION:
        JOptionPane.showMessageDialog(null,"I am glad you like Java Programming");
        break;
      case JOptionPane.NO_OPTION:
        JOptionPane.showMessageDialog(null, "You will like it if you come\n"
          + "to class and read the book");
        break;
      case JOptionPane.CANCEL_OPTION:
        JOptionPane.showMessageDialog(null, "Don't cancel,\nAnswer the question!");
        break;
      case JOptionPane.CLOSED_OPTION:
        JOptionPane.showMessageDialog(null, "Why did you close the window?\n"
          + "Answer the question!");
        break;
      default: JOptionPane.showMessageDialog(null,
        "Error: You should not have gotten here!");
    }
  }
}
```