Elementary Programming

CSE 114, Computer Science 1
SUNY Korea

Pradipta De
pradipta.de@sunykorea.ac.kr

Acknowledgement for the slides: Dr. Paul Fodor (CS Stony Brook)
Elementary programming

- To obtain the input from the console.
- To write simple programs with numerical expressions, other types (char, String), other programming conventions (spacing) and program design, running and debugging programs.
Variables

- In a program, data is stored in variables.
- Primitives store single pieces of data (ex: `char`)
  ```java
  char letter = 'A';
  ```
- Objects store multiple pieces of data (ex: `String`)
  ```java
  String text = "ABCDEFG";
  ```
- All Java variables must have a declared type.
- A variable’s type determines:
  - what kind of value the variable can hold
  - how much memory to reserve for that variable
Common Java Naming Conventions

- Variables & Method names start with **lowercase** letters: `x`, `toString`
- Classes start with **Uppercase** letters:
  - Ex: `public class Person { }`
- Variables and Class identifiers generally are **nouns**
- **Method** identifiers should be **verbs**
- Use Camel notation: `myVariable`, `MyClass`
- Although it is legal, do not begin with ‘_’ (underscore).
- Use descriptive names: `LinkedList`, `compareTo`

```java
area = PI * radius * radius;
```
Rules for Identifiers

- Should contain only letters, numbers, & ‘_’
- Cannot begin with a digit
- Uppercase and lowercase letters are considered different characters
  - Dummy and DUMMY and dummy are different identifiers
- $ is allowed, but only for special use → Do not use
- Examples:
  - Legal: myVariable, my_class, my4Var
  - Illegal: 4myVariable, my class, my!Var, @$myClass
Java’s Primitive Types

- **Integers (whole numbers)**
  - `byte`—1 byte (-128 to 127)
  - `short`—2 bytes (-32768 to 32767)
  - `int`—4 bytes (-2147483648 to 2147483647)
  - `long`—8 bytes (-9223372036854775808 to 9223372036854775807)

- **Real Numbers**
  - `float`—4 bytes
  - `double`—8 bytes

- **char**—2 bytes
  - stores a single character (Unicode 2)

- **boolean**—stores `true` or `false` (uses 1-bit or byte)
Variables

• Must be declared before being assigned values

```java
public void methodWithGoodDeclaration(){
    double salary;
    salary = 20000.0;
    System.out.println("Salary is "+ salary);
}
```

```java
public void methodWithBadDeclaration(){
    salary = 20000.0;
    double salary;
    System.out.println("Salary is "+ salary);
}
```
Variables

- **Must be initialized before being referenced**

```java
public void methodWithGoodReference() {
    double salary;
    salary = 20000.0;
    double raise = salary * 0.05; // 5% raise
    System.out.println("Raise is "+ raise);
}

public void methodWithBadReference() {
    double salary; // Salary has no value.
    double raise = salary * 0.05;
    System.out.println("Raise is "+ raise);
}
```
Variables

• A variable gets a value in an assignment statement
  
  \[
  \text{Variable} = \text{some}_\text{value} \text{ or an expression ;}
  \]

• Undefined Variables
  
  • Compiler error
  
  • If a variable does not occur on the left in an assignment statement before its use in an expression, then it is probably undefined
Variables

- **Should only be declared once**

```java
class Variables {
    public void methodWithGoodDeclaration() {
        double salary = 20000.0;
        System.out.println("Salary is "+ salary);
        salary = 60000.0;
        System.out.println("Salary is "+ salary);
    }

    public void methodWithBadDeclaration() {
        double salary = 50000.0;
        System.out.println("Salary is "+ salary);
        double salary = 60000.0; // Second declaration
        System.out.println("Salary is "+ salary);
    }
}
```
• Variables can be declared and initialized at once

```java
char yesChar = 'y';
String word = "Hello!";
double avg = 0.0, stdDev = 0.0;
char initial3 = 'T';
boolean completed = false;
```
Variables

- Variables can only be used inside the block { ...} or scope that they themselves are declared

```java
public void methodWithGoodScope()
{
    double x = 5.0;
    if (x > 0.0)
        System.out.println("x is " + x);
} // x is in scope here.

public void methodWithBadScope()
{
    double y = 100.0;
    if (y > 0.0)
    {
        double x = 5.0;
    }
    System.out.println("x " + x); // x is not in scope
}
Variables

- The Assignment Statement
  \[ \text{variable} = \text{expression}; \]
- What does it do?
  - Solves/evaluates expression first
  - Assigns resulting value to the variable
- Exercise: What’s the output?
  ```
  int x = 5;
  x = x + x + x + x + 10;
  System.out.print(x);
  ```
  ? 
  13
Variables

- The Assignment Statement
  
  ```java
  variable = expression;
  ```

- What does it do?
  - Solves/evaluates expression first
  - Assigns resulting value to the variable

- Exercise: What’s the output?

  ```java
  int x = 5;
  x = x + x + x + 10;
  System.out.print(x);
  ```

  25
• Assignment Compatibility:
  • The variable and expression should be the same type
    • if not, you may get a compiler error.
• Examples:

```java
int sumGrades, gradeX, gradeY;
gradeX = 1;
sumGrades = 1473;
sumGrades = 1472 + 1;
sumGrades = 1472 + gradeX;
sumGrades = true;       // ILLEGAL
```
Variables

- What about mixing numeric types?
- Are these assignment statements ok?
  
  ```java
  int x = 5;
  long y = x;
  double z = y;
  ```

- What about these?
  
  ```java
  double a = 6.5;
  long b = a;
  int c = b;
  ```

- byte < short < int < long < float < double

- No assigning big types to little types OR real types to integer types
Variables

• Type Casting as a type override
  • temporarily change a data type to another type
  • (type_name), example: (int)
  • no type casting to/from boolean

• Examples:
  
  ```java
  double myReal = 10.0;
  int badInt = myReal;
  int goodInt = (int)myReal;
  ```
Arithmetic Operators

+ Addition
- Subtraction
* Multiplication
/ Division
% Modulo/Remainder (integer operands only)

```java
int x = 5;
int y = 10;
int z = 2;
int num1 = (x + y) * z;
System.out.println(num1);
```

? 18
Arithmetic Operators

+    Addition
-    Subtraction
*    Multiplication
/    Division
%    Modulo/Remainder (integer operands only)

```java
int x = 5;
int y = 10;
int z = 2;
int num1 = (x + y) * z;
System.out.println(num1);  // Output: 30
```
Arithmetic Operators

- Multiplication (*) has higher precedence over addition (+).
  
  ```java
  int x = 5;
  int y = 10;
  int z = 2;
  int num1 = x + y * z;
  System.out.println(num1);
  
  int r2d2c3po = 3 * 4 + 5 / 6; // ?
  int r2d2c3po2 = (3 * (4 + 5))/ 6; // ?
  
  // Whenever in doubt, go with explicit use of parentheses.
  
  My Advice: avoid rules of precedence
  ```

- Whenever in doubt, go with explicit use of parentheses.

- My Advice: avoid rules of precedence
Arithmetic Operators

- Multiplication (*) has higher precedence over addition (+).
  ```java
  int x = 5;
  int y = 10;
  int z = 2;
  int num1 = x + y * z;
  System.out.println(num1);
  // 25
  ```

- Whenever in doubt, go with explicit use of parentheses.

- My Advice: avoid rules of precedence
  ```java
  int r2d2c3po = 3 * 4 + 5 / 6;  // 12
  int r2d2c3po2 = (3 * (4 + 5)) / 6;  // 4
  ```
Division

• Integer division:
  • $8/3 = 2$

• Double division:
  • $8.0/3.0 = 2.666666666666667$
  • $8.0/3 = 2.666666666666667$
  • $8/3.0 = 2.666666666666667$
Arithmetic Operators

- Division Operator - evaluate full expression first

```java
double average = 100.0/8.0;    //12.5
average = 100.0/8;             //12.5
average = 100/8;               //12.0
int sumGrades = 100/8;         //12
sumGrades = 100.0/8.0;         //ERROR
sumGrades = (int)100.0/8.0;    //ERROR
sumGrades = (int)(100.0/8.0);  //12
int fifty_percent = 50/100;    //0
double fiftyPercent = 50/100;   //0.0
fiftyPercent = 50.0/100.0;      //0.5
```
Arithmetic Operators

- The modulo/remainder % operator
- Produces division remainders

```java
int remainder = 100 % 8;
System.out.println(remainder);
```

?
Arithmetic Operators

- The modulo/remainder % operator
- Produces division remainders

```java
int remainder = 20 % 8;
System.out.println(remainder);  // 4
```
Arithmetic Operators

++ Increment by one
-- Decrement by one
+= Increment by specified amount
-= Decrement by specified amount
*=(Multiply by specified amount
/= Divide by specified amount

int x = 5, y = 15, z = 25;
x = x + 1;
y++;
z += 1;
System.out.println(x);
System.out.println(y);
System.out.println(z);
Arithmetic Operators

++    Increment by one
--    Decrement by one
+=    Increment by specified amount
-=    Decrement by specified amount
*=    Multiply by specified amount
/=    Divide by specified amount

```java
int x = 5, y = 15, z = 25;
x = x + 1;
y++;
z += 1;
System.out.println(x);  // 6
System.out.println(y);  // ?
System.out.println(z);  // ?
```
Arithmetic Operators

++  Increment by one
--  Decrement by one
+=  Increment by specified amount
-=  Decrement by specified amount
*=  Multiply by specified amount
/=  Divide by specified amount

```java
int x = 5, y = 15, z = 25;
x = x + 1;
y++;
z += 1;
System.out.println(x);  // 6
System.out.println(y);  // 16
System.out.println(z);  // ?
```
Arithmetic Operators

++  Increment by one
--  Decrement by one
+=  Increment by specified amount
-=  Decrement by specified amount
*=  Multiply by specified amount
/=  Divide by specified amount

int x = 5, y = 15, z = 25;
x = x + 1;
y++;
z += 1;

System.out.println(x);  // Output: 6
System.out.println(y);  // Output: 16
System.out.println(z);  // Output: 26
Increment and Decrement Operators

```c
int i = 10;
int newNum = 10 * i++; 
int newNum = 10 * i;  
i = i + 1;
```

Same effect as

```c
int newNum = 10 * i;
i = i + 1;
```

```c
int i = 10; 
int newNum = 10 * (i++); 
```

Same effect as

```c
i = i + 1;
int newNum = 10 * i;
```
Scientific Notation

- Floating-point literals can also be specified in scientific notation:
  - E (or e) represents an exponent and it can be either in lowercase or uppercase
  - Examples
    - $1.23456e+2 = 1.23456e2 = 123.456$
    - $1.23456e-2 = 0.0123456$
A class is a template or blueprint for objects

A program is defined by using one or more classes

```java
public class ClassName {
    public static void main(String[] args) {
        // ClassName PROGRAM’S POINT OF ENTRY
        // THIS PROGRAM’S INSTRUCTIONS
        // START HERE
    }
}
```
Methods

A method is a collection of statements that performs a sequence of operations

It is used by invoking a statement with arguments

```
System.out.println("Welcome to Java!");
```
The main Method

The main method provides the control of program flow.

- **ClassName** is executable because it has a main method
  - we can compile and then run it
- Not all classes require main methods
  - only those classes that initiate program execution require a main method

```java
public class ClassName {
    public static void main(String[] args) {
        ...
    }
}
```
/**
 * HelloWorld is a Java application
 * that simply displays "Hello World!" in the
 * Java console.
 */

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
        // Statement above displays "Hello World!"
    }
}

Our first program: HelloWorld.java
Computing the Area of a Circle:

```java
public class ComputeArea {
    public static void main(String[] args) {
        double radius; // Declare radius
        double area; // Declare area
        // Assign a radius
        radius = 20; // New value is radius
        // Compute area
        area = radius * radius * 3.14159;
        // Display results
        System.out.println("The area for the circle" +
                           + " of radius " + radius + " is " + area);
    }
}
```
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display results
        System.out.println("The area for the circle of radius " +
                radius + " is " + area);
    }
}
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display results
        System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
}
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display results
        System.out.println("The area for the circle of radius " +
                         radius + " is " + area);
    }
}
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display results
        System.out.println("The area for the circle of radius "+
                          radius + " is " + area);
    }
}
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

        // Compute area
        area = radius * radius * 3.14159;

        // Display results
        System.out.println("The area for the circle of radius "+
                           radius + " is "+ area);
    }
}
Reading Input from the Console

1. Create a Scanner object

   Scanner input = new Scanner(System.in);

2. Use the methods `next()`, `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain a `String`, `byte`, `short`, `int`, `long`, `float`, `double`, or `boolean` value. For example,

   System.out.print("Enter a double value: ");
   Scanner input = new Scanner(System.in);
   double d = input.nextDouble();

Scanner is in the Java package `java.util`
- start your program with:

   import java.util.Scanner;
Packages

• To make types easier to find and use, to avoid naming conflicts, and to control access, programmers bundle groups of related types into packages.

• The types that are part of the Java platform are members of various packages that bundle classes by function: fundamental classes are in \textit{java.lang}, classes for reading and writing (input and output) are in \textit{java.io}, and so on.

• You can put your types in packages too.
  • To create a package, you choose a name for the package and put a package statement with that name at the top of every source file that contains the types (e.g., classes, interfaces). In file Circle.java:

```java
package edu.stonybrook.cse114;
public class Circle {
    ...
}
```
Packages

- To use a public package member from outside its package, you must do one of the following:
  - Refer to the member by its fully qualified name
    ```java
    java.util.Scanner input =
    new java.util.Scanner(System.in);
    ```
  - Import the package member
    ```java
    import java.util.Scanner;
    ```
  - Import the member's entire package
    ```java
    import java.util.*;
    ```
• Packages appear to be hierarchical, but they are not.
  
  • Importing \texttt{java.awt.*} imports all of the types in the \texttt{java.awt} package, but it does not import \texttt{java.awt.color}, \texttt{java.awt.font}, or any other \texttt{java.awt.xxxx} packages.

  • If you plan to use the classes and other types in \texttt{java.awt.color} as well as those in \texttt{java.awt}, you must import both packages with all their files:
    
    \begin{verbatim}
    import java.awt.*;
    import java.awt.color.*;
    \end{verbatim}
Java Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles
- compact1
- compact2
- compact3

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.applet</td>
<td>Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.</td>
</tr>
<tr>
<td>java.awt</td>
<td>Contains all of the classes for creating user interfaces and for painting graphics and images.</td>
</tr>
<tr>
<td>java.awt.color</td>
<td>Provides classes for color spaces.</td>
</tr>
<tr>
<td>java.awt.datatransfer</td>
<td>Provides interfaces and classes for transferring data between and within applications.</td>
</tr>
<tr>
<td>java.awt.dnd</td>
<td>Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.</td>
</tr>
</tbody>
</table>
Constants

final datatype CONSTANTNAME = VALUE;

Examples:
final double PI = 3.14159;
final int SIZE = 3;
Character Data Type

```java
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
```

Four hexadecimal digits.

The increment and decrement operators can also be used on `char` variables to get the next or preceding Unicode character.

- the following statements display character b:

  ```java
  char ch = 'a';
  System.out.println(++ch);
  ```
Java characters use *Unicode UTF-16*

16-bit encoding

Unicode takes two bytes, preceded by `\u`, expressed in four hexadecimal numbers that run from `\u0000` to `\uFFFF`.

Unicode can represent $65535 + 1$ characters.

Unicode `\u03b1 \u03b2 \u03b3` for three Greek letters
### Escape Sequences for Special Characters

<table>
<thead>
<tr>
<th>Description</th>
<th>Escape Sequence</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>\b</td>
<td>\u0008</td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
<td>\u0009</td>
</tr>
<tr>
<td>Linefeed</td>
<td>\n</td>
<td>\u000A</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
<td>\u000D</td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
<td>\u005C</td>
</tr>
<tr>
<td>Single Quote</td>
<td>'</td>
<td>\u0027</td>
</tr>
<tr>
<td>Double Quote</td>
<td>&quot;</td>
<td>\u0022</td>
</tr>
</tbody>
</table>
Casting between char and Numeric Types

```c
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```
The char type only represents one character. To represent a string of characters, use the data type called String.

```java
String message = "Welcome to Java";
```

- **String** is a predefined class in the Java library just like the `System` class.
- The **String** type is **NOT** a primitive type.
- The **String** type is a *reference type*.
- **String variable** = a reference variable, which points to an object storing the value or actual text
More about Strings

• Each character is stored at an index:

```java
String sentence = "A statement";

012345678910
```

• The String class (from J2SE) has methods to process strings:

```java
System.out.println("charAt(6) is " + sentence.charAt(6));
System.out.println(sentence.toUpperCase());
System.out.println(sentence.substring(0,7) + sentence.substring(14));
```
Strings are immutable!

- There are no methods to change them once they have been created
- Assign the new one to the old variable

```java
String word = "Steven";
word = word.substring(0, 5);
```
String Concatenation

• “+” used for making a new string by concatenating strings

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B';
    // s1 becomes SupplementB
Useful String functions

- `charAt`, `equals`, `equalsIgnoreCase`, `compareTo`, `startsWith`, `endsWith`, `indexOf`, `lastIndexOf`, `replace`, `substring`, `toLowerCase`, `toUpperCase`, `trim`

- `s.equals(t)`
  - returns `true` if `s` and `t` have same letters and sequence
  - `false` otherwise
Special Characters

- \n  – newline
- \t  – tab
- \"  – quotation mark

Example: `print <img src="./pic.jpg" />`

```java
String s = "<img src="./pic.jpg" />";
System.out.print(s + "\n");
```
Design, Programming Style and Documentation

• **Design** = generalized steps of software engineering:
  1. Understand and define the problem
  2. Determine the required input and output
  3. Design an algorithm to solve the problem by computer
  4. Implement (code) the solution
  5. Debug and test the software
  6. Maintain and update the software

• **Programming Style and Documentation**
  • Appropriate Comments
  • Naming Conventions
  • Proper Indentation and Spacing Lines
  • Block Styles
ChangeMaker

Problem:
- you have to give someone change
- what coins do you give that person?

Requirements:
- takes user input
- displays the change breakdown as output
1. Understand and Define the Problem

- ask user for input
- US coins (quarter, dime, nickel, penny)
- max change: 99¢
- display coin output

- What’s involved?
  - interview users
  - What are their expectations?
  - What data do they need to access?
  - write a requirements analysis report
2. Determine Input and Output

- Typed input by user: amount of change requested (an integer between 1 and 99)
- Printed output:
  - Number of quarters given
  - Number of dimes given
  - Number of nickels given
  - Number of pennies given
3. Design an algorithm

- How many quarters?
  - subtract the number of quarters \( \times 25 \) from the total
- How many dimes?
  - subtract the number of dimes \( \times 10 \) from remaining total
- How many nickels?
  - subtract the number of nickels \( \times 5 \) from remaining total
- How many pennies?
  - the remaining total
3. Design an algorithm (cont.)

- Pseudocode: Use div and mod (remainder operator)

  User Inputs originalAmount
  numQuarters = originalAmount div 25
  remainder = originalAmount mod 25
  numDimes = remainder div 10
  remainder = remainder mod 10
  numNickels = remainder div 5
  remainder = remainder mod 5
  numPennies = remainder

  Output numQuarters
  Output numDimes
  Output numNickels
  Output numPennies
import java.util.Scanner;

public class ChangeMaker {
    public static void main(String[] args) {
        int change, rem, qs, ds, ns, ps;
        System.out.print("Input change amount (1-99): ");
        Scanner input = new Scanner(System.in);
        change = input.nextInt();
        qs = change / 25;
        rem = change % 25;
        ds = rem / 10;
        rem = rem % 10;
        ns = rem / 5;
        rem = rem % 5;
        ps = rem;
        System.out.print(qs + " quarters," + ds + " dimes, ");
        System.out.println(ns + " nickels and" + ps + " pennies");
    }
}
```java
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;
// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

5. Debug and test the software
Suppose amount is 11.56

```java
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
Suppose amount is 11.56

```java
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfOneQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

```java
remainingAmount 56
numberOfOneDollars 11
numberOfOneQuarters 2
```

numberOfOneQuarters assigned
Suppose amount is 11.56

```java
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```
Block Styles

Use end-of-line style for braces.

```java
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

Next-line style

End-of-line style
Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result
public class ShowSyntaxErrors {
    public static void main(String[] args) {
        i = 30;
        System.out.println(i + 4);
    }
}

public class ShowRuntimeErrors {  
    public static void main(String[] args) {
        int i = 1 / 0;
    }
}
public class ShowLogicErrors {
    // Determine if a number is between 1 and 100 inclusively
    public static void main(String[] args) {
        // Prompt the user to enter a number
        String input = JOptionPane.showInputDialog(null,
                "Please enter an integer:",
                "ShowLogicErrors", JOptionPane.QUESTION_MESSAGE);
        int number = Integer.parseInt(input);
        // Display the result
        System.out.println(
                "The number is between 1 and 100, inclusively? " +
                ((1 < number) && (number < 100)));

        System.exit(0);
    }
}
Debugging

• Logic errors are called *bugs*

• The process of finding and correcting errors is called debugging.

• Methods:
  • hand-trace the program (i.e., catch errors by reading the program),
  • insert print statements in order to show the values of the variables
  • for a large, complex program, the most effective approach for debugging is to use a *debugger utility*
Debugger

Debugger is a program that facilitates debugging. You can use a debugger to:

• Execute a single statement at a time.
• Trace into or stepping over a method.
• Set breakpoints.
• Display variables.
• Display call stack.
• Modify variables.
public void refreshDisplay(String option) {
    System.out.println("Option: "+option);
    if (option.equals("b41")) {
        bet = 1;
    } else if (option.equals("b42")) {
        bet = 2;
    } else if (option.equals("b43")) {
        bet = 3;
    } else if (option.equals("b44")) {
        bet = 4;
    } else if (option.equals("b51")) {
        b41.disable();
        b42.disable();
        b43.disable();
        b51.enable();
        b6.disable();
    } else if (option.equals("b52")) {
        // implement second step of baccarat
    }
    l42.setText("Bet: "+bet);
}
Our textbook provides two ways of obtaining input:

1. Using the Scanner class (console input)
2. Using JOptionPane input dialogs
Getting Input from Input Dialog Boxes

String input = JOptionPane.showInputDialog("Enter an input");
Getting Input from Input Dialog Boxes

String string = JOptionPane.showInputDialog(null, "Prompting Message", "Dialog Title", JOptionPane.QUESTION_MESSAGE);
Two Ways to Invoke the Method

- Several ways to use the `JOptionPane.showInputDialog` method:
  
  ```java
  String string =
  JOptionPane.showInputDialog(null, x, 
  y, JOptionPane.QUESTION_MESSAGE);
  ```
  
  where `x` is a string for the prompting message, and `y` is a string for the title of the input dialog box.

  ```java
  JOptionPane.showInputDialog(x);
  ```
  
  where `x` is a string for the prompting message.
Converting Strings to Integers

• The input returned from the input dialog box is a String
  • If you enter a numeric value such as 123, it returns “123”.
  • you have to convert a string into a number - use the static `parseInt` method in the `Integer` class:

```java
int intValue = Integer.parseInt(intString);
```

where `intString` is a numeric String (“123”)
Converting Strings to Doubles

Use the static `parseDouble` method in the `Double` class:

```java
double doubleValue = Double.parseDouble(doubleString);
```

where `doubleString` is a numeric String (“123.05”)