Elementary Programming

CSE 114, Computer Science 1

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Variables

- In a program, the variables store data
- There are 2 types of variables in Java (and most other modern programming languages):
  - Primitive type variables store single pieces of data:

```
int i = 1;
char letter = 'A';
```

  - Object or reference type variables store multiple pieces of data (ex: a **`String`** is a sequence of potentially multiple characters):

```
String text = "ABCDEFG";
```

Variables

- 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

```
char letter;
int i;
double area;
String s;
Object o;
```

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) – default (4321)
  - **`long`**–8 bytes (-9223372036854775808 to 9223372036854775807)
- Real Numbers
  - **`float`**–4 bytes (3.14159f)
  - **`double`**–8 bytes - default (3.141592)
- **`char`**–2 bytes
  - stores a single character (Unicode 2)
- **`boolean`**–stores **`true`** or **`false`** (uses 1-bit or byte)

Variables

- A variable gets a value in an assignment statement:

`Variable = some_value or`

`an expression ;`

Variables

- **A variable must be <u>declared before being assigned values:</u>**

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

Variables

- **Variables can be declared and initialized at once:**

```
char yesChar = 'y';
String word = "Hello!";
double avg = 0.0, stdDev = 0.0;
int i, j=0, k;
char initial3 = 'T';
boolean completed = false;
```

Variables

- **Local variable must be <u>initialized before being referenced:</u>**

```
public void methodWithGoodReference(){
  double salary = 20000.0; // GOOD
  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;
      // COMPILER ERROR: salary has no value
  System.out.println("Raise is " + raise);
}
```

Variables

- **A variable should only be declared once:**

```
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);
}                                          //COMPILER ERROR
```

Variables

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

```
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
}                              // COMPILER ERROR
```

Variables

- The Assignment Statement

```
variable = 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 + 10;
System.out.print(x);
```

?

Variables

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

```
int sumGrades, gradeX, gradeY;
gradeX = 1;
sumGrades = 1473;
sumGrades = 1472 + 1;
sumGrades = 1472 + gradeX;
sumGrades = true;  // ILLEGAL IN JAVA
              // COMPILER ERROR
```

Variables

- **<u>What about mixing numeric types?</u>**
- **Are these assignment statements ok?**

```
int x = 5;
long y = x;
double z = y;
```

- **What about these?**

```
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)**
  - **Examples:**

```
double myReal = 10.0;
int badInt = myReal; // Error
int goodInt = (int)myReal;//Good
```

  - **no type casting is allowed to/from boolean**

Arithmetic Operators

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

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

?

Arithmetic Operators

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

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

30

Arithmetic Operators

- Multiplication (*) has higher precedence over addition (+)

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

?

- **My Advice: avoid rules of precedence**
  - whenever in doubt, go with explicit use of parentheses.

```
int r2d2c3po = 3 * 4 + 5 / 6;
int r2d2c3po2 = (3 * (4 + 5))/ 6;
```

?

?

Division

- Integer division:
  - 8/3 = 2
- Double division:
  - 8.0/3.0 = 2.6666666666666667
  - 8.0/3 = 2.6666666666666667
  - 8/3.0 = 2.6666666666666667

Arithmetic Operators

- **Division operator (evaluate full expression first, then assignment):**

```
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 = 50/100.0;    //0.5
```

Arithmetic Operators

- The modulo/remainder % operator
  - Produces division remainders

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

?

Arithmetic Operators

- **The modulo/remainder % operator**
  - **Produces division remainders**

```
int remainder = 100 % 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**

```
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);
```

```
6
?
?
```

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);
```

```
6
16
?
```

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);
```

```
6
16
26
```

Increment and Decrement Operators

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

Same effect as

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

newNum is 100

i is 11

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

Same effect as

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

i is 11

newNum is 110

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
```

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
```

Scientific Notation

- Double values as 64-bit "double-precision" values, according to the IEEE 754 standard (https://en.wikipedia.org/wiki/IEEE_754-2008_revision):
  - Floating point numbers are represented internally as binary (base-2) fractions.
  - Most decimal fractions cannot be represented exactly as binary fractions, so in most cases the internal representation of a floating-point number is an approximation of the actual value.
  - In practice, the difference between the actual value and the represented value is very small and should not usually cause significant problems.

Classes

A program is defined by using one or more classes

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

A **class** is also a template or blueprint for **objects** (later)

Methods

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

```
public static int sum(int a, int b) {
    return a + b;
}
```

It is used by invoking a statement with arguments:

```
System.out.println( sum(5,6) );
```

The main Method

- The main method provides the control of program flow.

```
public class ClassName {
    public static void main(String[] args) {
        ...
    }
}
```

- ***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

HelloWorldApp.java

```
/**
* HelloWorldApp is a Java application
* that simply displays "Hello World!" in the
* Java console.
*/
public class HelloWorldApp {
  public static void main(String[] args) {
      System.out.println("Hello, World!");
      // Statement above displays "Hello, World!"
  }
}
```

- Computing the Area of a Circle:

```
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);
  }
}
```

Trace a Program Execution

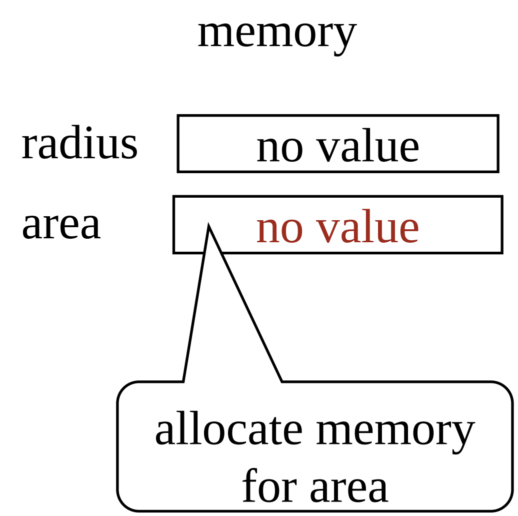
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // 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);
  }
}
```

allocate memory for radius

radius: no value

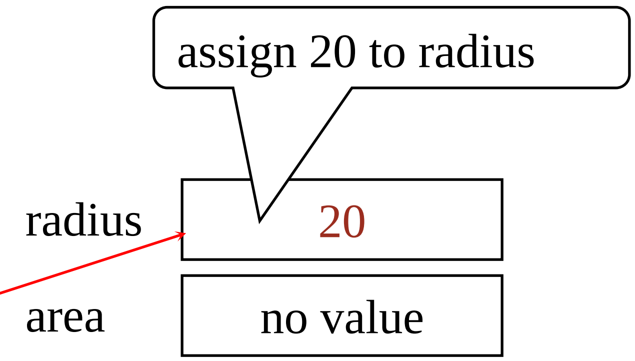
Trace a Program Execution

```
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);
  }
}
```



Trace a Program Execution

```
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);
  }
}
```



Trace a Program Execution

```
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);
  }
}
```

memory

radius	20
area	1256.636

compute area and assign it to variable area

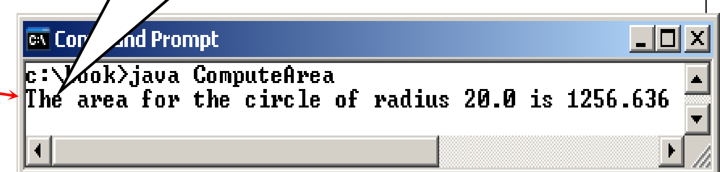
Trace a Program Execution

```
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);
  }
}
```

memory

radius	20
area	1256.636

print a message to the console



ChangeMaker.java

```java
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");
  }
}
```

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 *java.lang*, classes for reading and writing (input and output) are in *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:

```
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.util.Scanner input =
  new java.util.Scanner(System.in);
```

  - Import the package member

```
import java.util.Scanner;
```

  - Import the entire package

```
import java.util.*;
```

Packages

- Packages appear to be hierarchical, but they are not.
  - Importing **`java.awt.*`** imports all of the types in the java.awt package, but it does not import **`java.awt.color`**, **`java.awt.font`**, or any other **`java.awt.xxxx`** packages.
  - If you plan to use the classes and other types in **`java.awt.color`** as well as those in **`java.awt`**, you must import both packages with all their files:

```
import java.awt.*;
import java.awt.color.*;
```

Setting the CLASSPATH System Variable

- In Windows: **`set CLASSPATH=C:\users\george\java\classes`**
- In Unix-based OS:

```
%CLASSPATH=/home/george/java/classes;
export CLASSPATH
```

Constants

```
final datatype CONSTANTNAME = VALUE;
```

- Examples:

```
final double PI = 3.14159;
final int SIZE = 3;
```

Character Data Type

Four hexadecimal digits.

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

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**:

```
char ch = 'a';
System.out.println(++ch);
```

Unicode Format

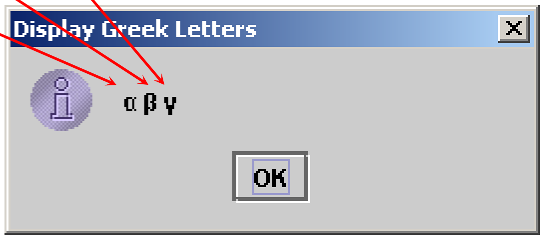
Java characters use *Unicode* UTF-16

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

Description	*Escape Sequence*	*Unicode*
Tab	`\t`	`\u0009`
Linefeed	`\n`	`\u000A`
Backslash	`\\`	`\u005C`
Single Quote	`\'`	`\u0027`
Double Quote	`\"`	`\u0022`

Casting between char and Numeric Types

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

char c = 97; // Same as char c = (char)97;
```

Software Development Process = 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

ChangeMaker

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

ChangeMaker

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

ChangeMaker

3. Design an algorithm
   - How many quarters?
     - subtract the number of quarters X 25 from the total
   - How many dimes?
     - subtract the number of dimes X 10 from remaining total
   - How many nickels?
     - subtract the number of nickels X 5 from remaining total
   - How many pennies?
     - the remaining total

ChangeMaker

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
```

4. Implement (code) the solution

```
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");
  }
}
```

5. Debug and test the software

Suppose amount is 11.56

```
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;
```

remainingAmount | 1156

remainingAmount initialized

Trace / Debug

Suppose amount is 11.56

```
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;
```

remainingAmount	1156
numberOfOneDollars	11

numberOfOneDollars assigned

Trace / Debug

Suppose amount is 11.56

```
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;
```

remainingAmount	56
numberOfOneDollars	11

remainingAmount updated

Trace / Debug

Suppose amount is 11.56

```
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;
```

remainingAmount	56
numberOfOneDollars	11
numberOfOneQuarters	2

numberOfOneQuarters assigned

Trace / Debug

Suppose amount is 11.56

```
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;
```

remainingAmount	6
numberOfOneDollars	11
numberOfQuarters	2

remainingAmount updated