Variables

• In a program, the variables store data

• 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

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

- There are 2 types of variables in Java (and most other modern programming languages):
  - **Primitive type** variables store single pieces of data:
    ```java
    int i = 1;    // i
    char letter = 'A';    // letter
    ```
  - **Object or reference type** variables store the reference (i.e., address) to an object that has multiple pieces of data (ex: a `String` is a sequence of potentially multiple characters):
    ```java
    String text = "ABCDEFG";
    ```
Java’s Primitive Types

- Integers (whole numbers)
  - `byte` – 1 byte (-128 to 127)
  - `short` – 2 bytes (-32,768 to 32,767)
  - `int` – 4 bytes (-2,147,483,648 to 2,147,483,647) – default for integer constants in the program
  - `long` – 8 bytes (-9223372036854775808 to 9223372036854775807)

- Real Numbers
  - `float` – 4 bytes
  - `double` – 8 bytes - default for real constants in the program

- `char` – 2 bytes
  - stores a single character (Unicode 2)

- `boolean` – stores `true` or `false` (uses 1-bit or byte)
Variable Assignment

• A variable gets a value in an assignment statement:

\[
\text{Variable} = \text{some\_value\_or\_an\_expression};
\]

Examples:

\[
\begin{align*}
\text{double} & \quad \text{salary}; \\
\text{salary} & \quad = \ 20000.0; \\
\text{char} & \quad \text{grade} = \ 'A'; \\
\text{String} & \quad \text{name} = \ "Paul";
\end{align*}
\]
Variables can be declared and initialized at once:

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

- The Assignment Statement

  \[ \text{variable} = \text{expression}; \]

  What does it do?

  1. **Solves/evaluates expression first!**
  2. **Assigns resulting value to the variable!**

- Exercise: What’s the output?

  ```java
  int x = 5;
  x = x + x + 10;
  System.out.print(x); // 20
  ```
Variables

- A variable **must be declared before being assigned values:**

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

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

- A **local** variable must be **initialized before being used**:

  ```java
  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;
      // SYNTAX ERROR because salary has no value
      System.out.println("Raise is " + raise);
  }
  ```
Variables

- A variable should only be declared once in one block:

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

```java
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);
}                         //Syntax ERROR
```
Variables

• Local 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){
        x = 5.0;
    }
    System.out.println("x " + 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); // SYNTAX ERROR // x is not in scope
```

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Variables

- **Assignment Compatibility:**
  - The variable and expression should be of compatible types.
  - If not, you may get a compiler error.

- Examples:

```c
int sumGrades, gradeX, gradeY;
gradeX = 1; // GOOD
sumGrades = 1473; // GOOD
sumGrades = 1472 + 1; // GOOD
sumGrades = 1472 + gradeX; // GOOD
sumGrades = true; // SYNTAX ERROR
sumGrades = 5.4; // Syntax Error
```
Variables

- **What about mixing numeric types?**
- These assignment statements are ok:
  
  ```java
  int x = 5;
  long y = x;
  double z = y;
  ```
  
  because: `byte < short < int < long < float < double`

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

- No assigning big type values to little type variables OR real type values to integer type variables
Variables

• **Type Casting**: change a data type value to another type (sometimes with some loss):

  \[(\text{type\_name})\text{expression}\]

• **Example**:

  ```
  double myReal = 10.5;
  int goodInt = (int)myReal;//Good
  // goodInt is 10
  ```

• **No type casting is allowed to/from boolean**
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);  // 30
```
Division

• Integer division (->quotient):
  • $8/3 = 2$

• Double division:
  • $8.0/3.0 = 2.6666666666666667$
  • $8.0/3 = 2.6666666666666667$
  • $8/3.0 = 2.6666666666666667$
Division

- 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 = 50.0/100.0;   // 0.5
  ```
Rules of precedence

- PEMDAS order of operations:
  - Multiplication and division (*/+) have higher precedence over addition and subtraction (+-)

```java
int x = 5;
int y = 10;
int z = 2;
int num1 = x + y * z;
System.out.println(num1);  // 25
```

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

```java
int r2d2c3po = 3 * 4 + 5 / 6;  // 12
int r2d2c3po2 = (3 * (4 + 5)) / 6;  // 4
```
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
*=
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);
Pre and Post Increment and Decrement Operators

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

Same effect as
i = i + 1;
int newNum = 10 * i;

i=11
newNum = 110

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

Same effect as
int newNum = 10 * i;
i = i + 1;

i=11
newNum = 100
Pre and Post Increment

```java
int i = 10;
i = ++i + i++;
     // (i=11) 11 + 11 (i=12) = 22
System.out.println(i); // 22

int i = 10;
i = i++ + i++;
     // 10 (i=11) + 11 (i=12) = 21
System.out.println(i); // 21

int y = 5;
y -= y++ - --y;
     // y = 5 - (5 (y=6) - (y=5) 5) = 5 - (5 - 5) = 5 - 0 = 5
System.out.println(y); // 5

• Notes:
y -= val;
y = y - val;
```
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
“double-precision” values

- double values are represented internally as 64-bit “double-precision” values, according to the IEEE 754 standard ([https://en.wikipedia.org/wiki/IEEE_754-2008_revision](https://en.wikipedia.org/wiki/IEEE_754-2008_revision)):
  - That is, floating point numbers are represented internally as binary (base-2) fractions.
  - But 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.

```java
System.out.println(1 - 0.1 - 0.1 - 0.1);
0.700000001
```
Constants

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

- Examples:
  ```java
  final double PI = 3.14159;
  final int SIZE; // assignment can be later
  SIZE = 3; // GOOD
  SIZE = 4; // ILLEGAL if changed again
  ```

- Convention (i.e., style): UPPERCASE letters are used for constants (FORTRAN did not have constants, so developers used uppercase only to communicate that the identifier is a constant)
char letter = 'A';
char numChar = '4';
Unicode

- Java characters use Unicode UTF-16 bit encoding
- chars can be assigned Unicode codes:
  ```java
  char letter = '\u0041'; // Unicode for 'A'
  char numChar = '\u0034'; // Unicode for '4'
  ```
  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
Character Data Type

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);
```
## Escape Sequences for Special Characters

<table>
<thead>
<tr>
<th>Description</th>
<th>Escape Sequence</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tab</td>
<td>\t</td>
<td>\u0009</td>
</tr>
<tr>
<td>Linefeed</td>
<td>\n</td>
<td>\u000A</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

```java
int i = 'a'; // Same as int i = (int)'a';
char c = 97; // Same as char c = (char)97;
```
Classes

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
    }
}
```

A **class** is also a template or blueprint for **objects** (we will see that later in Objects and Classes)
Methods

A method is a collection of statements that performs a sequence of operations. It is used by invoking a statement with arguments:

```java
System.out.println("Welcome to Java!");
```
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
/**
 * 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!"
    }
}
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);
    }
}

allocate memory for radius
no value

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area; // Allocate memory for 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);
    }
}
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,"
                        + 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 `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, `nextBoolean()` or `next()` to obtain a byte, short, int, long, float, double, boolean or String (up to the first white space) 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
    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

• 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:
    ```java
    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:
  ```bash
  %CLASSPATH=/home/george/java/classes;
  export CLASSPATH
  ```
Software engineering

- Software engineering waterfall model:
  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
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 the minimum number of coins (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 maximum number of quarters $\times 25c$ from the total

- How many dimes?
  - subtract the maximum number of dimes $\times 10c$ from remaining total

- How many nickels?
  - subtract the maximum number of nickels $\times 5c$ 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.println("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," + ns + " nickels and" + ps + " pennies");
    }
}
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;
```
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;
```
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;
```
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;
```
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;
```