Elementary Programming

CSE 114: Introduction to Object-Oriented Programming

Paul Fodor

Stony Brook University

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

Contents

- Identifiers
- Variables
  - Declaring Variables and Data Types (Java's Primitive Types)
  - Assignments and Assignment Compatibility
- Type Casting
- Arithmetic Operators
  - Pre and Post Increment and Decrement Operators
- Scientific Notation and "double-precision" values
- Constants
- Character Data Type and Unicode
- Classes, Methods and the main Method
  - HelloWorld.java, ComputeArea.java, ChangeMaker.java
- Reading Input from the Console and Packages in Java

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

Identifiers

- What's an Application Programming Interface (API)?
  - a library of code **identifiers**/names to use
- What are identifiers/names used for?
  - For Variables, Classes, and Methods
  - They come from 2 sources:
    - the Oracle (or someone else's) Java API
    - your own variables, classes and methods
  - Identifiers (Names) – Why name them?
    - they are your data and commands, and you'll need to reference them elsewhere in your program

```
int myVariable = 5; // Declaration
myVariable = myVariable + 1; // Using the variable
```


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

Rules for Identifiers

- Should contain only letters, numbers, & '_'
  - '$' is allowed, but only for special use
- Cannot begin with a digit!
- Although it is legal, do not begin with '_' (underscore)
- Uppercase and lowercase letters are considered to be different characters (Java is case-sensitive)
- Examples:
  - Legal: **`myVariable, my_class, my4Var`**
  - Illegal: **`4myVariable, my class, my!Var, @#$myClass`**

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

Common Java Naming Conventions

- Variables & Methods start with lower case letters: **`radius`**, **`getRadius`**
- Classes start with upper case letters: **`Circle`**
- Variables and Class identifiers should generally be nouns: **`radius`**, **`Circle`**
- Method identifiers should be verb phrases: **`getRadius`**
- Use Camel notation: **`GeometricObject`**, **`getRadius`**
- Use descriptive names: **`Circle`**, **`radius`**, **`area`**

```
area = PI * radius * radius;
```


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

Variables

- In a program, the variables store data
- All Java variables must have a declared type

```
type variable;
```

  - A variable's type determines:
    - what kind of value the variable can hold
    - how much memory to reserve for that variable

```
int i;
long l;
double area;
char letter;
String s;
```


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

Data Types

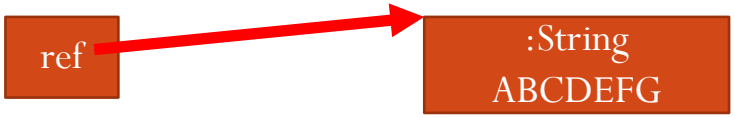
- There are 2 categories of types in Java (and most other modern programming languages):
  - Primitive type variables store single pieces of data:

```
int i = 1;
```

```
char letter = 'A';
```

  - Object or reference type variables store the reference (i.e., address, aka. pointer) to an object that may have multiple pieces of data and *instance* methods (ex: a **`String`** is a sequence of multiple characters):

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



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

Java's 8 Primitive Types

- Integers (whole numbers):
  - **`byte`**–represented in 1 byte (8 bits) (-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`**–represented in 2 bytes to store a single character (Unicode2/UTF16 variable encoding)
- **`boolean`**–stores **`true`** or **`false`** (uses 1-bit)

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

Assignments

- A variable gets a value in an assignment statement:

```
Variable = some_value or
           an expression ;
```

Examples:

```
double salary;
salary = 20000.0;
char grade;
grade = 'A';
```


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

Assignments

- Variables can be declared and initialized at once:

```
char yesChar = 'y';
String word = "Hello!";
boolean completed = false;
```

- We can declare and (optionally) assign multiple variables in one statement:

```
double total, count=0, avg = 0.0,
  stdDev, his = 1.0;
```


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

Assignments

- The Assignment Statement

```
variable = expression;
```

What does it do?

1. **First: Solves/evaluates expression!**
2. Assigns resulting value to the left variable!

- Exercise: What's the output if the same variable appear to the left and right of an assignment?

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

20

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

Variables

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

```
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);
}
```


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

Variables

- A local variable must be initialized before being used:

```
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);
}
```


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

Variables

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

```
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; //Syntax ERROR
  System.out.println("Salary is " + salary);
 }
```


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

Scope of Variables

- The *scope* of a variable is the part of the program where the variable can be referenced.
  - In Java, the scope of a local variable starts from its declaration and continues to the end of the block that contains the variable

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

Variables

- **Local variables can only be used from their declaration until the end of the block where they were declared**

```
public void methodWithGoodScope(){
  double x = 5.0;
  if (x > 0.0){ // x is in scope here
      x = 6.0;  // including in inner blocks
  }
  System.out.println("x " + x); // x is still in scope here
}
public void methodWithBadScope(){
  double y = 100.0;
  if (y > 0.0) {
      double x = 5.0;
  } // no more x
  System.out.println("x " + x); // SYNTAX ERROR
}                               // x is not in scope
```


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

Variables

- A nested block cannot redefine a local variable:

```
public static void method() {
   int x = 1;
   {
      int x = 0; // Syntax error
   }
}
```


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

Variables

- Static variables declared in a class:

```
public class Test {
   public static int maxUsers = 100;
   public static void main(String[] args){
      System.out.println(maxUsers);
   }
}
```


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

Variables

- The scope of static variables is the entire class:

```
public class Test {
   public static void main(String[] args){
      System.out.println(maxUsers);
   }
   public static int maxUsers = 100;
}
// Bad style, but CORRECT
```


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

Compatibility

- **Assignment Compatibility:**
  - The expression should be of compatible type with the variable
    - if not, you may get a compiler error.
  - Examples:

```
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
```


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

Assignment Compatibility

- **What about mixing numeric types?**
- These assignment statements are ok:

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

because: byte < short < int < long < float < double

- What about these?

```
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

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

Assignment Compatibility

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

  (type_name)expression

  - Example:

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

  - No type casting is allowed to/from boolean

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

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

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

Division

- Integer division:
  - 8/3 = 2 (the quotient)
- Double division (if at least an operand is a double):
  - 8.0/3.0 = 2.6666666666666667
  - 8.0/3 = 2.6666666666666667
  - 8/3.0 = 2.6666666666666667

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

Division

- **Division examples (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
```


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

Rules of precedence

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

```
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**.

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

12

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

4

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

Arithmetic Operators

- The modulo/remainder % operator
  - Produces division remainders

```
int remainder = 10 % 6;
System.out.println(remainder);
```

```
4
```


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

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
```


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

Pre and Post Increment and Decrement Operators

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

Same effect as

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

Results in: i=11

newNum = 110

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

Same effect as

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

Results in: newNum = 100

i=11

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

Pre and Post Increment

```
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;` **IS** `y = y - val;`

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

Scientific Notation

- Floating-point literals can also be specified in scientific notation:
  - E (or e) represents an exponent of the base and it can be either in lowercase or uppercase
  - Examples

```
1.23456e+2 = 1.23456e2 = 123.456
1.23456e-2 = 0.0123456
```


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

“double-precision” values

- **`double`** values are represented internally according to the IEEE 754 standard (https://en.wikipedia.org/wiki/IEEE_754-2008_revision):
  - That is, real numbers are represented internally as sums of binary (base-2) fractions/negative powers of 2 (e.g., $0.5 = 2^{-1}$ $0.75 = 2^{-1} + 2^{-2}$).
    - But many/most decimal fractions (e.g, 1/10=0.1) cannot be represented exactly as binary fractions, so in many/most cases the internal representation of a real number is an approximation of the actual value.

```
System.out.println(1 - 0.1 - 0.1 - 0.1);
0.70000001
```


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

Constants

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

- Examples:

```
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 (because FORTRAN did not have constants, so developers used uppercase only to communicate that the identifier is a constant)

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

Character Data Type

```
char letter = 'A';
char numChar = '4';
```


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

Character Data Type

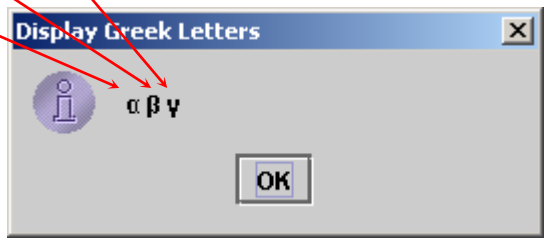
- Java characters use *Unicode* UTF-16 bit encoding
- chars can be assigned Unicode codes:

```
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`.

- Examples:

  Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



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

Casting between char and Numeric Types

```
int i = 'a'; // Same as int i = (int)'a';
   // i is 97
char c = (char)97;
// This is also allowed in Java:
char c = 97; // Same as char c = (char)97;
   // c is 'a'
```


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

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**:

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


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

Escape Sequences for Special Characters

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

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

Classes

A program is defined by using one or more classes

```
public class ClassName {
      // implementation
}
```

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

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

Methods

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

```
public static void print(String arg) {
  // implementation
}
```

-It is used by invoking the method with arguments.

```
System.out.print("Welcome to Java!");
```


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

The main Method

- The main method executes when we run a program:

```
public class ClassName {
    public static void main(String[] args) {
        // Execution POINT OF ENTRY

    }
}
```

  - ***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

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

Example programs: HelloWorld.java

```
/**
 HelloWorld.java 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!"
  }
}
```


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

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);
  }
}
```


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

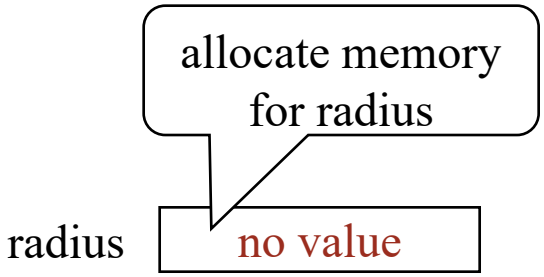
Trace a Program Execution

```
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);
  }
}
```



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

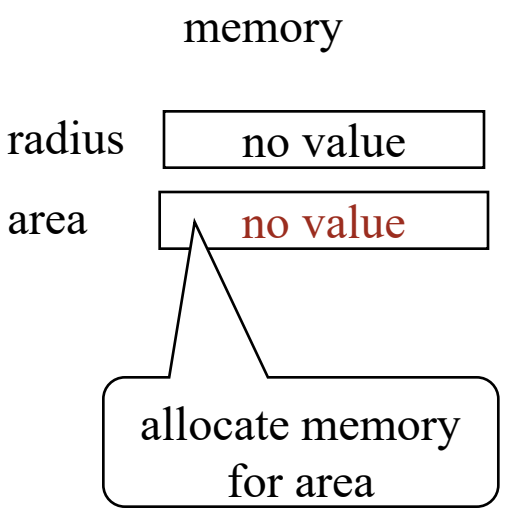
Trace a Program Execution

```
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);
  }
}
```



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

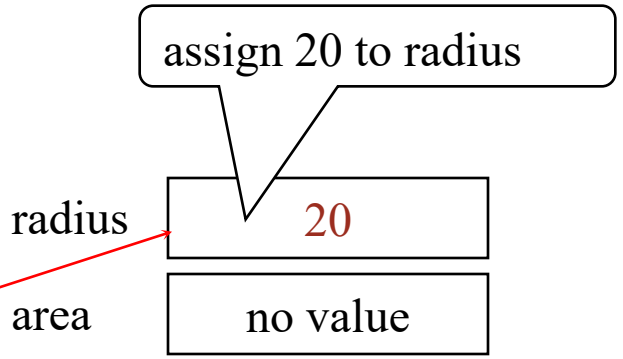
Trace a Program Execution

```
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);
  }
}
```



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

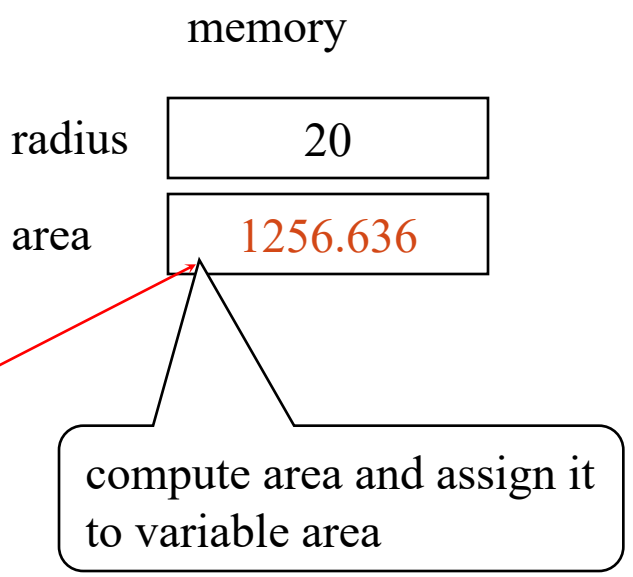
Trace a Program Execution

```
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);
  }
}
```



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

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

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

ChangeMaker.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");
  }
}
```


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

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;
```


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

Packages in Java

- To avoid naming conflicts and to control access, programmers bundle groups of related classes into packages.
- The classes 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.util* and *java.io*, and so on.
  - You can put your classes in packages too.
    - To create a package, you put a package statement with the name at the top of *the source file*.

```
package stonybrook.cse;
public class Circle {
    ...
}
```


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

Packages in Java

- To use a class from outside its package, you must do one of the following:
  - Import the package class:

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

  - Import the entire package:

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

  - Refer to the member by its fully qualified name every time you use it:

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


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

Packages in Java

- 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 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.*;
```


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

Extending ChangeMaker

```
import java.util.Scanner;

public class ChangeMaker {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    // read a price, e.g., $73.28
    // read an amount, e.g., $100
    // compute the change=amount-price, e.g., $100-$73.29=$26.72
    // use bills and coins to cover that change
    System.out.print("Input the price: $");
    double price = input.nextDouble();
    System.out.print("Input the paid amount: $");
    double amount = input.nextDouble();

    double change = amount - price;
    System.out.println("Change: $" + change);

    int rem = (int)(change * 100);

    int hundreds = rem / 10000;
    rem = rem % 10000;
    if(hundreds > 0)
      System.out.println(hundreds + " x $100 bills");
```


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

```
int fifties = rem / 5000;
rem = rem % 5000;
if(fifties > 0)
  System.out.println(fifties + " x $50 bills");

int twenties = rem / 2000;
rem = rem % 2000;
if(twenties > 0)
  System.out.println(twenties + " x $20 bills");

int tens = rem / 1000;
rem = rem % 1000;
if(tens > 0)
  System.out.println(tens + " x $10 bills");

int fives = rem / 500;
rem = rem % 500;
if(fives > 0)
  System.out.println(fives + " x $5 bills");

int ones = rem / 100;
rem = rem % 100;
if(ones > 0)
  System.out.println(ones + " x $1 bills");
```


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

```
    int qs = rem / 25;
    rem = rem % 25;
    if(qs > 0)
      System.out.println(qs + " x 25c");

    int ds = rem / 10;
    rem = rem % 10;
    if(ds > 0)
      System.out.println(ds + " x 10c");

    int ns = rem / 5;
    rem = rem % 5;
    if(ns > 0)
      System.out.println(ns + " x 5c");

    if(rem > 0)
      System.out.println(rem + " x 1c");

  }
}
```


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