Fundamental Data Types

Advanced Programming

ICOM 4015

Lecture 4

Reading: Java Concepts Chapter 4
Lecture Goals

• To understand integer and floating-point numbers
• To recognize the limitations of the numeric types
• To become aware of causes for overflow and roundoff errors
• To understand the proper use of constants

Continued…
Lecture Goals

• To write arithmetic expressions in Java
• To use the String type to define and manipulate character strings
• To learn how to read program input and produce formatted output
Number Types

- **int**: integers, no fractional part
  
  1, -4, 0

- **double**: floating-point numbers (double precision)
  
  0.5, -3.11111, 4.3E24, 1E-14
Number Types

• A numeric computation overflows if the result falls outside the range for the number type

```java
int n = 1000000;
System.out.println(n * n); // prints -727379968
```

• Java: 8 primitive types, including four integer types and two floating point types
## Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td>The integer type, with range (-2,147,483,648 \ldots 2,147,483,647)</td>
<td>4 bytes</td>
</tr>
<tr>
<td><code>byte</code></td>
<td>The type describing a single byte, with range (-128 \ldots 127)</td>
<td>1 byte</td>
</tr>
<tr>
<td><code>short</code></td>
<td>The short integer type, with range (-32768 \ldots 32767)</td>
<td>2 bytes</td>
</tr>
<tr>
<td><code>long</code></td>
<td>The long integer type, with range (9,223,372,036,854,775,808 \ldots ) (-9,223,372,036,854,775,807)</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

*Continued...*
## Primitive Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>The double-precision floating-point type, with a range of about ( \pm 10^{308} ) and about 15 significant decimal digits</td>
<td>8 bytes</td>
</tr>
<tr>
<td>float</td>
<td>The single-precision floating-point type, with a range of about ( \pm 10^{38} ) and about 7 significant decimal digits</td>
<td>4 bytes</td>
</tr>
<tr>
<td>char</td>
<td>The character type, representing code units in the Unicode encoding scheme</td>
<td>2 bytes</td>
</tr>
<tr>
<td>boolean</td>
<td>The type with the two truth values false and true</td>
<td>1 byte</td>
</tr>
</tbody>
</table>
Number Types: Floating-point Types

- Rounding errors occur when an exact conversion between numbers is not possible

```java
double f = 4.35;
System.out.println(100 * f); // prints 434.99999999999994
```

- Java: Illegal to assign a floating-point expression to an integer variable

```java
double balance = 13.75;
int dollars = balance; // Error
```
Number Types: Floating-point Types

- **Casts**: used to convert a value to a different type
  
  ```java
  int dollars = (int) balance; // OK
  ```

  Cast discards fractional part.

- **Math.round**: converts a floating-point number to nearest integer
  
  ```java
  long rounded = Math.round(balance); // if balance is 13.75, then
  // rounded is set to 14
  ```
Syntax 4.1: Cast

\[(typeName)\ expression\]

**Example:**
\[(int) (balance \ast 100)\]

**Purpose:**
To convert an expression to a different type
Self Check

1. Which are the most commonly used number types in Java?

2. When does the cast \((\text{long}) x\) yield a different result from the call \(\text{Math.round}(x)\)?

3. How do you round the \(\text{double}\) value \(x\) to the nearest \(\text{int}\) value, assuming that you know that it is less than \(2 \cdot 10^9\)?
Answers

- int and double
- When the fractional part of $x$ is $\geq 0.5$
- By using a cast: `(int) Math.round(x)`
Constants: `final`

- A `final` variable is a constant
- Once its value has been set, it cannot be changed
- Named constants make programs easier to read and maintain
- Convention: use all-uppercase names for constants

```java
final double QUARTER_VALUE = 0.25;
final double DIME_VALUE = 0.1;
final double NICKEL_VALUE = 0.05;
final double PENNY_VALUE = 0.01;
payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE
   + nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
```
Constants: static final

- If constant values are needed in several methods, declare them together with the instance fields of a class and tag them as static and final
- Give static final constants public access to enable other classes to use them

```java
public class Math {
    . . .
    public static final double E = 2.7182818284590452354;
    public static final double PI = 3.14159265358979323846;
}

double circumference = Math.PI * diameter;
```
Syntax 4.2: Constant Definition

In a method:
final typeName variableName = expression ;

In a class:
accessSpecifier static final typeName variableName = expression ;

Example:
final double NICKEL_VALUE = 0.05;
public static final double LITERS_PER_GALLON = 3.785;

Purpose:
To define a constant in a method or a class
A cash register totals up sales and computes change due.

```java
public class CashRegister {
    /**
     * Constructs a cash register with no money in it.
     */
    public CashRegister() {
        purchase = 0;
        payment = 0;
    }
}
```

Continued...
File **CashRegister.java**

```java
15:    /**
16:     * Records the purchase price of an item.
17:     * @param amount the price of the purchased item
18:     */
19:    public void recordPurchase(double amount)
20:    {
21:        purchase = purchase + amount;
22:    }
23:
24:    /**
25:     * Enters the payment received from the customer.
26:     * @param dollars the number of dollars in the payment
27:     * @param quarters the number of quarters in the payment
28:     * @param dimes the number of dimes in the payment
29:     * @param nickels the number of nickels in the payment
30:     * @param pennies the number of pennies in the payment
31:     */
```
```java
32:   public void enterPayment(int dollars, int quarters,
33:                  int dimes, int nickels, int pennies)
34:   {
35:       payment = dollars + quarters * QUARTER_VALUE
36:           + dimes * DIME_VALUE
37:           + nickels * NICKEL_VALUE + pennies
38:           * PENNY_VALUE;
39:   }
40:   /*
41:      Computes the change due and resets the machine for
42:          the next customer.
43:      @return the change due to the customer
44: */
```
public double giveChange()
{
    double change = payment - purchase;
    purchase = 0;
    payment = 0;
    return change;
}

public static final double QUARTER_VALUE = 0.25;
public static final double DIME_VALUE = 0.1;
public static final double NICKEL_VALUE = 0.05;
public static final double PENNY_VALUE = 0.01;
private double purchase;
private double payment;
File CashRegisterTester.java

```java
/**
 * This class tests the CashRegister class.
 */
public class CashRegisterTester {
    public static void main(String[] args) {
        CashRegister register = new CashRegister();
        register.recordPurchase(0.75);
        register.recordPurchase(1.50);
        register.enterPayment(2, 0, 5, 0, 0);
        System.out.print("Change=");
        System.out.println(register.giveChange());
    }
}
Continued...
```
File `CashRegisterTester.java`

```java
16: register.recordPurchase(2.25);
17: register.recordPurchase(19.25);
18: register.enterPayment(23, 2, 0, 0, 0);
19: System.out.print("Change=");
20: System.out.println(register.giveChange());
21: }
22: }
```

**Output**

Change=0.25
Change=2.0
Self Check

1. What is the difference between the following two statements?

   final double CM_PER_INCH = 2.54;

   and

   public static final double CM_PER_INCH = 2.54;

2. What is wrong with the following statement?

   double circumference = 3.14 * diameter;
Answers

1. The first definition is used inside a method, the second inside a class

2. (1) You should use a named constant, not the "magic number" 3.14
   (2) 3.14 is not an accurate representation of π
Assignment, Increment, and Decrement

- Assignment is not the same as mathematical equality:
  \[ \text{items} = \text{items} + 1; \]
- \text{items}++ is the same as \text{items} = \text{items} + 1
- \text{items}-- subtracts 1 from \text{items}
Assignment, Increment and Decrement

Figure 1: Incrementing a Variable
Self Check

1. What is the meaning of the following statement?
   \[\text{balance = balance + amount;}\]

1. What is the value of \(n\) after the following sequence of statements?
   \[n--;\]
   \[n++;\]
   \[n++;\]
   \[n--;\]
Answers

1. The statement adds the \textit{amount} value to the \textit{balance} variable

2. One less than it was before
**Arithmetic Operations**

- `/` is the division operator
- If both arguments are integers, the result is an integer. The remainder is discarded.
  - $7.0 / 4$ yields $1.75$
  - $7 / 4$ yields $1$
- Get the remainder with `%` (pronounced "modulo")
  - $7 \% 4$ is $3$
Arithmetic Operations

```java
final int PENNIES_PER_NICKEL = 5;
final int PENNIES_PER_DIME = 10;
final int PENNIES_PER_QUARTER = 25;
final int PENNIES_PER_DOLLAR = 100;
// Compute total value in pennies
int total = dollars * PENNIES_PER_DOLLAR + quarters * PENNIES_PER_QUARTER
+ nickels * PENNIES_PER_NICKEL + dimes * PENNIES_PER_DIME
+ pennies;
// Use integer division to convert to dollars, cents
int dollars = total / PENNIES_PER_DOLLAR;
int cents = total % PENNIES_PER_DOLLAR;
```
The **Math** class

- **Math class**: contains methods like `sqrt` and `pow`
- **To compute** $x^n$, **you write** `Math.pow(x, n)`
- **However, to compute** $x^2$ **it is significantly** more efficient simply to compute $x \times x$
- **To take the square root of a number, use the** `Math.sqrt;` **for example,** `Math.sqrt(x)`
The Math class

• In Java,

\[ \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

can be represented as

\[ (-b + \text{Math.sqrt}(b * b - 4 * a * c)) / (2 * a) \]
## Mathematical Methods in Java

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.sqrt(x)</td>
<td>square root</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td>power $x^y$</td>
</tr>
<tr>
<td>Math.exp(x)</td>
<td>$e^x$</td>
</tr>
<tr>
<td>Math.log(x)</td>
<td>natural log</td>
</tr>
<tr>
<td>Math.sin(x), Math.cos(x), Math.tan(x)</td>
<td>sine, cosine, tangent ($x$ in radian)</td>
</tr>
<tr>
<td>Math.round(x)</td>
<td>closest integer to $x$</td>
</tr>
<tr>
<td>Math.min(x, y), Math.max(x, y)</td>
<td>minimum, maximum</td>
</tr>
</tbody>
</table>
Analyzing an Expression

Figure 3: Analyzing an Expression
Self Check

1. What is the value of $\frac{1729}{100}$? Of $1729 \mod 100$?

2. Why doesn't the following statement compute the average of $s1$, $s2$, and $s3$?

   ```java
   double average = s1 + s2 + s3 / 3; // Error
   ```

3. What is the value of

   ```java
   Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))
   ```

   in mathematical notation?
Answers

1. 17 and 29

2. Only $s_3$ is divided by 3. To get the correct result, use parentheses. Moreover, if $s_1$, $s_2$, and $s_3$ are integers, you must divide by 3.0 to avoid integer division:

$$(s_1 + s_2 + s_3) / 3.0$$

3. $\sqrt{x^2 + y^2}$
Calling Static Methods

- A static method does not operate on an object

```java
double x = 4;
double root = x.sqrt(); // Error
```

- Static methods are defined inside classes

- Naming convention: Classes start with an uppercase letter; objects start with a lowercase letter
**Syntax 4.3: Static Method Call**

```
ClassName. methodName(parameters)
```

**Example:**
```
Math.sqrt(4)
```

**Purpose:**
To invoke a static method (a method that does not operate on an object) and supply its parameters
Self Check

1. Why can't you call \( x.\text{pow}(y) \) to compute \( x^y \)?

2. Is the call `System.out.println(4)` a static method call?
Answers

1. **x** is a number, not an object, and you cannot invoke methods on numbers

2. No—the **println** method is called on the object **System.out**
Strings

• A string is a sequence of characters

• Strings are objects of the `String` class

• String constants:
  
  "Hello, World!"

• String variables:

  ```java
  String message = "Hello, World!";
  ```

• String length:

  ```java
  int n = message.length();
  ```

• Empty string:

  ""
Concatenation

• **Use the + operator:**

```java
String name = "Dave";
String message = "Hello, " + name;
// message is "Hello, Dave"
```

• **If one of the arguments of the + operator is a string, the other is converted to a string**

```java
String a = "Agent";
int n = 7;
String bond = a + n; // bond is Agent7
```
Concatenation in Print Statements

- **Useful to reduce the number of `System.out.print` instructions**

  ```java
  System.out.print("The total is ");
  System.out.println(total);
  ```

  versus

  ```java
  System.out.println("The total is " + total);
  ```
Converting between Strings and Numbers

- Convert to number:
  ```java
  int n = Integer.parseInt(str);
  double x = Double.parseDouble(str);
  ```

- Convert to string:
  ```java
  String str = "" + n;
  str = Integer.toString(n);
  ```
Substrings

- Supply start and “past the end” position
- First position is at 0

```java
String greeting = "Hello, World!";
String sub = greeting.substring(0, 5); // sub is "Hello"
```

Figure 3: String Positions

Continued...
Substrings

- Substring length is “past the end” - start

Figure 4: Extracting a Substring
Self Check

1. Assuming the String variable \( s \) holds the value "Agent", what is the effect of the assignment \( s = s + s.length() \)?

2. Assuming the String variable \( \text{river} \) holds the value "Mississippi", what is the value of \( \text{river.substring}(1, 2) \)? Of \( \text{river.substring}(2, \text{river.length()} - 3) \)?
Answers

1. \texttt{s} is set to the string \texttt{Agent5}

2. The strings \texttt{"i"} and \texttt{"ssissi"}
International Alphabets

Figure 5:
A German Keyboard
International Alphabets

Figure 6:
The Thai Alphabet
### CLASSIC SOUPS

<table>
<thead>
<tr>
<th>No.</th>
<th>Item</th>
<th>Sm.</th>
<th>Lg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>House Chicken Soup (Chicken, Celery,</td>
<td>1.50</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>Potato, Onion, Carrot)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Chicken Rice Soup</td>
<td>1.85</td>
<td>3.25</td>
</tr>
<tr>
<td>59</td>
<td>Chicken Noodle Soup</td>
<td>1.85</td>
<td>3.25</td>
</tr>
<tr>
<td>60</td>
<td>Cantonese Wonton Soup</td>
<td>1.50</td>
<td>2.75</td>
</tr>
<tr>
<td>61</td>
<td>Tomato Clear Egg Drop Soup</td>
<td>1.65</td>
<td>2.95</td>
</tr>
<tr>
<td>62</td>
<td>Regular Wonton Soup</td>
<td>1.10</td>
<td>2.10</td>
</tr>
<tr>
<td>63</td>
<td>Hot &amp; Sour Soup</td>
<td>1.10</td>
<td>2.10</td>
</tr>
<tr>
<td>64</td>
<td>Egg Drop Soup</td>
<td>1.10</td>
<td>2.10</td>
</tr>
<tr>
<td>65</td>
<td>Egg Drop Wonton Mix</td>
<td>1.10</td>
<td>2.10</td>
</tr>
<tr>
<td>66</td>
<td>Tofu Vegetable Soup</td>
<td>NA</td>
<td>3.50</td>
</tr>
<tr>
<td>67</td>
<td>Chicken Corn Cream Soup</td>
<td>NA</td>
<td>3.50</td>
</tr>
<tr>
<td>68</td>
<td>Crab Meat Corn Cream Soup</td>
<td>NA</td>
<td>3.50</td>
</tr>
<tr>
<td>69</td>
<td>Seafood Soup</td>
<td>NA</td>
<td>3.50</td>
</tr>
</tbody>
</table>

**Figure 7:**
A Menu with Chinese Characters

Fall 2006  Slides adapted from Java Concepts companion slides
Reading Input

- `System.in` has minimal set of features—it can only read one byte at a time

- In Java 5.0, `Scanner` class was added to read keyboard input in a convenient manner
  ```java
  Scanner in = new Scanner(System.in);
  System.out.print("Enter quantity: ");
  int quantity = in.nextInt();
  ```

- `nextDouble` reads a double

- `nextLine` reads a line (until user hits Enter)

- `nextWord` reads a word (until any white space)
import java.util.Scanner;

/**
 * This class tests console input.
 */

public class InputTester
{

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        CashRegister register = new CashRegister();

        System.out.print("Enter price: ");
        double price = in.nextDouble();
        register.recordPurchase(price);
    }
}
File `InputTester.java`

```java
18:     System.out.print("Enter dollars: ");
19:     int dollars = in.nextInt();
20:     System.out.print("Enter quarters: ");
21:     int quarters = in.nextInt();
22:     System.out.print("Enter dimes: ");
23:     int dimes = in.nextInt();
24:     System.out.print("Enter nickels: ");
25:     int nickels = in.nextInt();
26:     System.out.print("Enter pennies: ");
27:     int pennies = in.nextInt();
28:     register.enterPayment(dollars, quarters, dimes, nickels, pennies);
29: 
30:     System.out.print("Your change is ");
31:     System.out.println(register.giveChange());
32: }
33: }
```

Continued...
File InputTester.java

Output

Enter price: 7.55
Enter dollars: 10
Enter quarters: 2
Enter dimes: 1
Enter nickels: 0
Enter pennies: 0
Your change is 3.05
Reading Input from a Dialog Box

Figure 8:
An Input Dialog Box
Reading Input From a Dialog Box

- String input = JOptionPane.showMessageDialog(prompt)

- Convert strings to numbers if necessary:
  \[
  \text{int count} = \text{Integer.parseInt}(\text{input});
  \]

- Conversion throws an exception if user doesn't supply a number—see chapter 15

- Add `System.exit(0)` to the main method of any program that uses `JOptionPane`
Self Check

1. Why can't input be read directly from `System.in`?

2. Suppose `in` is a `Scanner` object that reads from `System.in`, and your program calls:
   ```java
   String name = in.next();
   ```
   What is the value of `name` if the user enters `John Q. Public`?
Answers

1. The class only has a method to read a single byte. It would be very tedious to form characters, strings, and numbers from those bytes.

2. The value is "John". The next method reads the next word.