Chapter Four: Fundamental Data Types
Chapter 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
- 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

- 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>int</td>
<td>The integer type, with range -2,147,483,648 \ldots 2,147,483,647</td>
<td>4 bytes</td>
</tr>
<tr>
<td>byte</td>
<td>The type describing a single byte, with range -128 \ldots 127</td>
<td>1 byte</td>
</tr>
<tr>
<td>short</td>
<td>The short integer type, with range -32768 \ldots 32767</td>
<td>2 bytes</td>
</tr>
<tr>
<td>long</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>
<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 bit</td>
</tr>
</tbody>
</table>
• 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
```

• 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, // rounded is set to 14
```
Syntax 4.1 Cast

\[(\text{typeName}) \text{ expression}\]

**Example:**

\[(\text{int}) (\text{balance} * 100)\]

**Purpose:**
To convert an expression to a different type.
Which are the most commonly used number types in Java?

**Answer:** int and double
When does the cast \((\text{long}) \ x\) yield a different result from the call \(\text{Math.round}(x)\)?

**Answer:** When the fractional part of \(x\) is \(\geq 0.5\)
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;
```
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;
    }

    /** Records the purchase price of an item. */
    public void recordPurchase(double amount) {
        purchase = purchase + amount;
    }
}
```
** Enter the payment received from the customer.  
@param dollars the number of dollars in the payment  
@param quarters the number of quarters in the payment  
@param dimes the number of dimes in the payment  
@param nickels the number of nickels in the payment  
@param pennies the number of pennies in the payment  
*/

```java
public void enterPayment(int dollars, int quarters,  
int dimes, int nickels, int pennies)
{
    payment = dollars + quarters * QUARTER_VALUE + dimes * DIME_VALUE  
+ nickels * NICKEL_VALUE + pennies * PENNY_VALUE;
}
```

** Computes the change due and resets the machine for the next customer.  
@return the change due to the customer  
*/

class CashRegister {
    public double giveChange()
    {
        Continued
45:     double change = payment - purchase;
46:     purchase = 0;
47:     payment = 0;
48:     return change;
49: }
50: }
51:     public static final double QUARTER_VALUE = 0.25;
52:     public static final double DIME_VALUE = 0.1;
53:     public static final double NICKEL_VALUE = 0.05;
54:     public static final double PENNY_VALUE = 0.01;
55: }
56:     private double purchase;
57:     private double payment;
58: }

/**
 * 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());
        System.out.println("Expected: 0.25");

        register.recordPurchase(2.25);
        register.recordPurchase(19.25);
        register.enterPayment(23, 2, 0, 0, 0);
        System.out.print("Change: ");
        System.out.println(register.giveChange());
        System.out.println("Expected: 2.0");
    }
}
Output:
Change: 0.25
Expected: 0.25
Change: 2.0
Expected: 2.0
• items = items + 1; \ new value of “items” is the old \ value of “items” plus 1
• items++ is the same as items = items + 1
• items-- subtracts 1 from items
Assignment, Increment, and Decrement

Figure 1
Incrementing a Variable
What is the meaning of the following statement?

\[ \text{balance} = \text{balance} + \text{amount}; \]

**Answer:** The statement adds the \text{amount} value to the \text{balance} variable.
What is the value of \( n \) after the following sequence of statements?

\[
\begin{align*}
n & --; \\
n & ++; \\
n & --;
\end{align*}
\]

\textbf{Answer:} 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

  \[ 7.0 \div 4 \text{ yields } 1.75 \]
  \[ 7 \div 4 \text{ yields } 1 \]

• Get the remainder with \( \% \) (pronounced "modulo")

  \[ 7 \% 4 \text{ is } 3 \]
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;
Static fields and methods

- Static fields/methods belong to classes
  - (non-static fields/methods belong to objects)
- Naming convention: Classes start with an uppercase letter; objects start with a lowercase letter

Math \ \ Math is a class
System.out \ \ out is a static object in the System \ \ class

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

...(elsewhere in your code) ...
double circumference = Math.PI * diameter;
```
**Syntax 4.3 Static Method Call**

`ClassName.methodName(parameters)`

**Example:**

`Math.sqrt(4)`

**Purpose:**
To invoke a static method and supply its parameters.

Note: Math is a class, not an object.
Is the call `System.out.println(4)` a static method call?

**Answer:** No – the `println` method is called on the object `System.out`. 
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)
- In Java, $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$ can be represented as $(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)$
## Mathematical Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Returns</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 radians)</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

\[
\frac{-b + \text{Math.sqrt}(b \times b - 4 \times a \times c))}{2 \times a}
\]

\[
\left\{ \begin{array}{c}
b^2 \\
4ac \\
2a
\end{array} \right.
\]

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

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

**Figure 2** Analyzing an Expression
Self Check 4.8

What is the value of $\frac{1729}{100}$? Of $1729\% \ 100$?

**Answer:** 17 and 29
Why doesn't the following statement compute the average of \( s_1, s_2, \) and \( s_3 \)?

\[
\text{double average = } s_1 + s_2 + s_3 / 3; \quad // \quad \text{Error}
\]

**Answer:** 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:

\[
(s_1 + s_2 + s_3) / 3.0
\]
Self Check 4.10

What is the value of $\text{Math.sqrt(Math.pow(x, 2) + Math.pow(y, 2))}$ in mathematical notation?

Answer: $\sqrt{x^2 + y^2}$
Strings

- A string is a sequence of characters
- Strings are objects of the String class
- String constants:
  "Hello, World!"
- String variables:
  String message = "Hello, World!";
- String length:
  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(x);
  ```

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

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

- Supply start and “past the end” position

- First position is at 0

Figure 3  String Positions

Continued
Substring length is “past the end” - start

Figure 4 Extracting a Substring
Self Check 4.13

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

**Answer:** `s` is set to the string `Agent5`
Assuming the String variable river holds the value "Mississippi ", what is the value of river.substring(1, 2)? Of river.substring (2, river.length() - 3)?

**Answer:** The strings "i" and "ssissi"
**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 program simulates a transaction in which a user pays for an item and receives change.
   */
public class CashRegisterSimulator {
    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);
        System.out.print("Enter dollars: ");
        int dollars = in.nextInt();
    }
}
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

An Input Dialog Box
Reading Input From a Dialog Box

• String input = JOptionPane.showInputDialog(prompt)

• Convert strings to numbers if necessary:
  int count = Integer.parseInt(input);

• Conversion throws an exception if user doesn't supply a number – see chapter 11

• Add System.exit(0) to the main method of any program that uses JOptionPane
Why shouldn't input be read directly from `System.in`?

**Answer:** 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.