CHAPTER 4

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 round-off 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

Number types

`int`: integers, no fractional part

`1, -4, 0`

`double`: floating-point numbers

`0.5, -3.11111, 4.3E24, 1E-14`

Java has 8 primitive types:

- four integer types including `int`
- two floating point types including `double`
- the character type `char`
- the truth-value type `boolean`

Floating-point types

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

```
double f = 4.35;
System.out.println(100 * f);
// prints 434.9999999999994
```

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

```
double balance = 13.75;
int dollars = balance; // Error
```

- Casts: Used to convert a value to a different type

```
int dollars = (int) balance; // OK
```

Cast discards fractional part.

Floating-point types (cont.)

Math.round converts a floating-point number to nearest integer

```
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 * 100)

Purpose:

To convert an expression to a different type.

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

```
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

```
public class Math
{
   . . .
   public static final double E =
   2.718281828459045235 4;
   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;
```

Purpose:
To define a constant in a method or a class.

The CashRegister class

```
public class CashRegister
{
  public CashRegister()
  {
    purchase = 0;
    payment = 0;
  }

  public void recordPurchase(double amount)
  {
    purchase = purchase + amount;
  }
```

The CashRegister class (cont.)

```
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;
}

public double giveChange()
{
  double change = payment - purchase;
  purchase = 0;
  payment = 0;
  return change;
}
```

The CashRegister class (cont.)

```
  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;
}
```

The CashRegisterTester 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");
   }
}
```

Arithmetic operations: Division

`/` is the division operator

If both arguments are integers, the result is an integer.

The remainder is discarded

```
7.0/4 = 1.75
7 / 4 = 1
```

Get the remainder with `%` (pronounced "modulo")

```
7%4 = 3
```

Mathematical Functions

`Math.sqrt(x)`	square root
`Math.pow(x, y)`	power x^y
`Math.exp(x)`	e^x
`Math.log(x)`	natural log
`Math.sin(x)`, `Math.cos(x)`, `Math.tan(x)`	sine, cosine, tangent
`Math.round(x)`	closest integer to x

Analyzing an Expression

```
(-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a)
```

b^2 $4ac$ $2a$

$b^2 - 4ac$

$\sqrt{b^2 - 4ac}$

$-b + \sqrt{b^2 - 4ac}$

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

Calling static methods

- A `static` method does not operate on an object

```
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

  `Math`

  `System.out`

Syntax 4.3: Static Method Call

ClassName .*methodName*(*parameters*)

Example:

```
Math.sqrt(4)
```

Purpose:

To invoke a static method (a method that doesn't operate on an object) and supply its parameters.

Strings

String constants:

```
"Carl"
```

String variables:

```
String name = "Carl";
```

String length:

```
int n = name.length();
```

Concatenation

```
String fname = "Harry";
String lname = "Hacker";
String name = fname + lname;
```

`name` is `"HarryHacker"`

If one operand of `+` is a string,
the other is converted to a string:

```
String a = "Agent";
String name = a + 7;
```

`name` is `"Agent7"`

Converting between Strings and Numbers

- Convert to number:

```
int n = Integer.parseInt(str);
double x = Double.parseDouble(str);
```

- Convert to string:

```
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
  ```

Substrings (cont.)

- `String greeting = "Hello, World!";`
  `String sub =greeting.substring(7, 12);`
  `// sub is "World"`
- Substring length is "past the end" – start

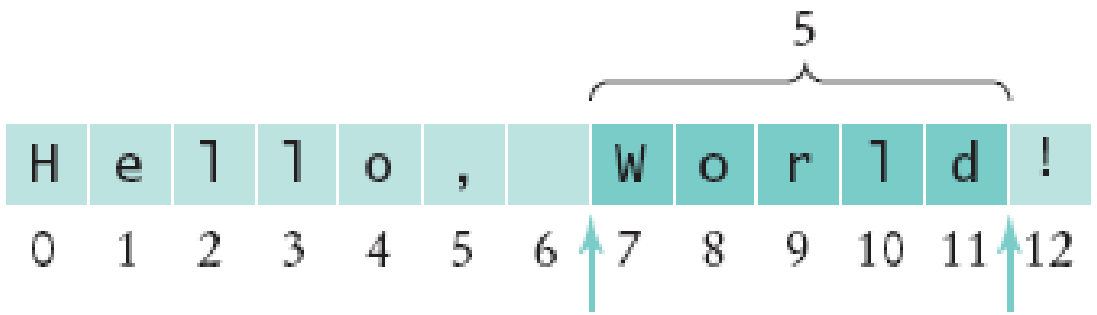

Figure 4 Extracting a Substring

Characters

`char`: character type—a single character

Character constants use single quotes:
`'A', 'B', 'a'`

`'A'` is not the same as `"A"`

`charAt` method gets character from a string
`"Hello".charAt(0)` is `'H'`

Reading input

- In Java 5.0, the `Scanner` class was added to read keyboard input in a convenient manner
-
```
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)

The CashRegisterSimulator class

```
import java.util.Scanner;

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);
```

The CashRegisterSimulator class (cont.)

```
System.out.print("Enter dollars: ");
int dollars = in.nextInt();
System.out.print("Enter quarters: ");
int quarters = in.nextInt();
System.out.print("Enter dimes: ");
int dimes = in.nextInt();
System.out.print("Enter nickels: ");
int nickels = in.nextInt();
System.out.print("Enter pennies: ");
int pennies = in.nextInt();
```

The CashRegisterSimulator class (cont.)

```
      register.enterPayment(dollars, quarters,
      dimes, nickels, pennies);
      System.out.print("Your change: ");
      System.out.println(register.giveChange());
   }
}
```