Chapter outline

- loop techniques
  - cumulative sum
  - for loop loops

- conditional execution
  - the if statement and the if/else statement
  - relational expressions
  - nested if/else statements

- subtleties of conditional execution
  - object equality
  - factoring if/else code
  - text processing
  - methods with conditional execution: returning return values

Adding many numbers

Consider this code to read and add three values:
```java
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int sum1 = console.nextInt();
System.out.print("Type a number: ");
int sum2 = console.nextInt();
System.out.print("Type a number: ");
int sum = sum1 + sum2 + sum;
System.out.println("The sum is " + sum);
```

A cumulative sum

- The variables num1, num2, and num3 are unnecessary:
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("Type a number: ");
  int sum = console.nextInt();
  System.out.print("Type a number: ");
  sum += console.nextInt();
  System.out.print("Type a number: ");
  sum += console.nextInt();
  System.out.println("The sum is " + sum);
  ```

- cumulative sum: A variable that keeps a sum-in-progress and is updated many times until the task of summing is finished.
  - The variable `sum` in the above code is a cumulative sum.

Failed cumulative sum loop

- How could we modify the code to sum 100 numbers?
  - Creating 100 copies of the same code would be redundant.

- An incorrect solution:
  ```java
  Scanner console = new Scanner(System.in);
  for (int i = 1; i <= 100; i++)
    int sum = 0;
    System.out.print("Type a number: ");
    sum = console.nextInt();
    sum += sum;
  System.out.println("The sum is " + sum);
  ```

- The scope of `sum` is outside the for loop, so the last line of code fails to compile.
Fixed cumulative sum loop

A corrected version of the sum loop code:

```java
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++)
    System.out.print("Type a number: ");
    sum += console.nextInt();
System.out.println("The sum is "+sum);
```

The key idea:

- Cumulative sum variables must always be declared outside the loops that update them, so that they will continue to hold after the loop is finished.

User-guided cumulative sum

User input can control the number of loop repetitions:

```java
Scanner console = new Scanner(System.in);
System.out.print("How many numbers to add: ");
int count = console.nextInt();
int sum = 0;
for (int i = 1; i <= count; i++)
    System.out.print("Type a number: ");
    sum += console.nextInt();
System.out.println("The sum is "+sum);
```

An example output:

- How many numbers to add: 100
  Type a number: 2
  Type a number: 4
  Type a number: 6
  ... (continue adding numbers)
  The sum is: 1000

Variation: cumulative product

The same idea can be used with other operators, such as multiplication which produces a cumulative product:

```java
Scanner console = new Scanner(System.in);
int product = 1;
for (int i = 1; i <= exponent; i++)
    product *= 2;
System.out.println("2 raised to the "+exponent+" is "+product);
```

Exercises:

- Change the above code so that it also prompts for the base, instead of always using 2.
- Change the above code into a method which accepts a base `a` and exponent `b` as parameters and returns `a^b`.

Cumulative sum question

Write a program that reads input of the number of hours two employees have worked and displays each employee’s total and the overall total hours.

- The company doesn’t pay overtime, so cap any day at 8 hours.

Example log of execution:

1. Employee 1: How many hours?
   Hours: 10
   Hours: 12
   Employee 1’s total hours = 22

2. Employee 2: How many hours?
   Hours: 8
   Employee 2’s total hours = 8
   Total hours for both = 30

Cumulative sum answer

// Computes the total hours worked by two employees.
// The company doesn’t pay overtime, so cap any day at 8 hours.
// Returns the total hours worked by the employees.
public static int processEmployee(Scanner console, int number) {
    System.out.print("Employee "+number+": How many days? ");
    int days = console.nextInt();
    int total = 0;
    for (int i = 1; i <= days; i++)
        System.out.print("Hour "+i+":");
    System.out.println("Total ");
    return total;
}

Cumulative sum answer 2

---

// Reads hours information about one employee with the given number.
// Returns the total hours worked by the employee.
public static int processEmployee(Scanner console, int number) {
    System.out.print("Employee "+number+": How many days? ");
    int days = console.nextInt();
    int total = 0;
    for (int i = 1; i <= days; i++)
        System.out.print("Hour "+i+":");
    System.out.println("Total ");
    return total;
}
**Fencepost loops**

reading: 4.1

---

**The fencepost problem**

- Problem: Write a static method named `printNumbers` that prints each number from 1 to a given maximum, separated by commas.

For example, the method call:

```java
printNumbers(5)
```

should print:

```
1, 2, 3, 4, 5
```

---

**Flawed solution 1**

- A flawed solution:
  ```java
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(i + " ");
          System.out.println(); // to end the line of output
      }
  }
  ```

- Output from `printNumbers(5)`: 1, 2, 3, 4, 5.

---

**Flawed solution 2**

- Another flawed solution:
  ```java
  public static void printNumbers(int max) {
      for (int i = 1; i <= max; i++) {
          System.out.print(i + " ");
          System.out.println(); // to end the line of output
      }
  }
  ```

- Output from `printNumbers(5)`: 1, 2, 3, 4, 5.

---

**Fencepost analogy**

- We print `n` numbers but need only `n - 1` commas.
- This problem is similar to the task of building a fence with lengths of wire separated by posts, often called a fencepost problem.
- If we repeatedly place a post and wire, the last post will have an extra dangling wire.

- A flawed algorithm:
  ```java
  for (length of fence) {
      place some post.
      place same wire.
  }
  ```

---

**Fencepost loop**

- The solution is to add an extra statement outside the loop that places the initial "post."
  ```java
  for (length of fence - 1) {
      place same wire.
      place some post.
  }
  ```
**Fencepost method solution**

- A version of `printNumbers` that works:
  ```java
  public static void printNumbers(int max) {
    System.out.print("");
    for (int i = 2; i <= max; i++) {
      System.out.print(" + " + i);
    }
    System.out.println(" = ");
  }
  var output = printNumbers(5);
  System.out.println(output);
  ```

**Fencepost question**

- Write a method named `printFactors` that, when given a number, prints its factors in the following format:
  ```java
  if (number % i == 0) {
    System.out.print(i + " ");
  }
  ```

**Fencepost question**

- Write a Java program that reads a base and a maximum power and prints all of the powers of the given base up to that max, separated by commas.

  ```java
  Base: 2
  Max exponent: 5
  The first 5 powers of 2 are:
  2, 4, 8, 16, 32, 64, 128, 256, 512
  ```

**The if statement**

- **if statement**: A Java statement that executes a block of statements only if a certain condition is true.
  - If the condition is not true, the block of statements is skipped.
  - General syntax:
    ```java
    if (<condition>) {
      <statement>;
      <statement>;
      <statement>;
    }
    ```
  - Example:
    ```java
    double gpa = Integer.parseInt(input.nextLine());
    if (gpa >= 3.0) {
      System.out.println("Your application is accepted.");
    }
    ```

**if statement flow diagram**

```
<table>
<thead>
<tr>
<th>is the test true?</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
</tr>
<tr>
<td>execute statement after if statement</td>
</tr>
<tr>
<td>no</td>
</tr>
</tbody>
</table>
```
The if/else statement

- **if/else statement**: A Java statement that executes one block of statements if a certain condition is true, and a second block of statements if it is false.
  - **General syntax**:
    ```java
    if (<condition>) {
        <statement1>
    } else {
        <statement2>
    }
    ```
  - **Example**:
    ```java
    double gpa = console.nextDouble();
    if (gpa >= 2.0) {
        System.out.println("Welcome to Mars University!");
    } else {
        System.out.println("Your application is denied.");
    }
    ```

Relational expressions

- The `<condition>` used in an if or if/else statement is the same kind seen in a for loop.
- The conditions are actually of type boolean, seen in Ch. 5.
- These conditions are called relational expressions and use one of the following six relational operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equal</td>
<td>x == y</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
<td>x != y</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>x &gt; y</td>
<td>true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>x &gt;= y</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>x &lt; y</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>x &lt;= y</td>
<td>true</td>
</tr>
</tbody>
</table>

Evaluating rel. expressions

- Relational operators have lower precedence than math operators.
- **Examples**:
  - `5 + 7 <= 10 <= 3 * (7 - 1)`
    
  - `35 >= 3 * 30 = 30 <= true`

Loops with if/else

- Loops can be used with if/else statements:
  ```java
  int negativex = 0;
  int x = 0;
  int next = console.readInt();
  if (next == -1) {
      System.out.println("'next' is negativex;
          } else {
      negativex = x;
          }
      
  public static void printOdd(int max)
  
  System.out.println("'next' is odd";
  if (x <= max) {
      System.out.println("'next' is even";
      System.out.println("'next' is odd";
  }
  ```

if/else question

- Write code to read a number from the user and print whether it is even or odd using an if/else statement.
- **Example execution**:
  ```java
  Type a number: 8
  Your number is even
  ```
  ```java
  Type a number: 9
  Your number is odd
  ```
Nested if/else statements

- **nested if/else statement**: A chain of if/else that chooses between outcomes using many conditions.
- General syntax:
  ```java
  if (<condition1>):
      <statement1>
  else if (<condition2>):
      <statement2>
  else:
      <statement3>
  ```
- Example:
  ```java
  if (value > 0):
      System.out.println("Positive");
  else if (value < 0):
      System.out.println("Negative");
  else:
      System.out.println("Zero");
  ```

Nested if/else variations

- A nested if/else can end with an if or an else.
  - If ends with else, one of the code paths must be taken.
  - If ends with if, the program might not execute any path.
- Example ending with if:
  ```java
  if (value == 0):
      System.out.println("Zero");
  else if (value == 1):
      System.out.println("You win the gold medal!");
  else if (value == 2):
      System.out.println("You win a silver medal!");
  else if (value == 3):
      System.out.println("You earned a bronze medal.");
  ```
- Are there any cases where this code will not print a message?
- How could we modify it to print a message to non-medals?

Nested if/else flow diagram

```java
if (<condition>):
  <statement1>
else if (<condition>):
  <statement2>
else:
  <statement3>
```

Nested if/else/if diagram

```java
if (<condition>):
  <statement1>
else if (<condition>):
  <statement2>
else:
  <statement3>
```

Sequential if diagram

```java
if (<condition>):
  <statement1>
if (<condition>):
  <statement2>
if (<condition>):
  <statement3>
```
Which nested if/else to use?

- Which if/else construct is most appropriate?
  - Reading the user's GPA and printing whether the student is on the dean's list (3.6 to 4.0) or honor roll (3.5 to 3.8).
  - Printing whether a number is even or odd.
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
  - Printing a user's grade of A, B, C, D, or F based on their percentage in the course.

Which nested if/else answers

- Which if/else construct is most appropriate?
  - Reading the user's GPA and printing whether the student is on the dean's list (3.6 to 4.0) or honor roll (3.5 to 3.8).
  - Printing whether a number is even or odd.
  - Simple if/else
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
  - *nested if / else if / else
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
  - *sequential if / if / if
  - Printing a user's grade of A, B, C, D, or F based on their percentage in the course.
  - *nested if / else if / else if / else

How to comment: if/else

- Comments shouldn't describe the condition being tested.
  - Instead, describe why you are performing that test, or what you intend to do based on its result.

  Example:
  ```java
  // Simple example:
  if (num > 20) {
      System.out.println("The number is greater than 20");
  } else {
      System.out.println("The number is less than or equal to 20");
  }
  ```

  Better example:
  ```java
  // Better example:
  if (num > 20) {
      System.out.println("The number is greater than 20");
  } else { // Use a more descriptive comment.
      System.out.println("The number is less than or equal to 20");
  }
  ```

Math.max/min vs. if/else

- Many if/else statements that choose the larger or smaller of 2 numbers can be replaced by a call to Math.max or Math.min.
  ```java
  int x = 10, y = 20;
  int z = Math.min(x, y);
  ```

- Simple if/else:
  ```java
  int x = Math.max(a, b);
  ```

Subtleties of conditional execution reading: 4.3

- When comparing numbers, use the correct comparison operator. For example,
  ```java
  if (x < y) {
      System.out.println("x is less than y");
  }
  ```

- Use the correct comparison operator when comparing objects.
  ```java
  if (x.equals(y)) {
      System.out.println("x is equal to y");
  }
  ```
Comparing objects

- Relational operators such as `<=` and `>=` only behave correctly on primitive values.
  - The `==` operator on strings often evaluates to `false` even when the strings have the same letters in them.

  **Example (incorrect):**
  ```java
  Scanner console = new Scanner(System.in);
  System.out.print("Enter your name ": n);
  String name = console.nextLine();
  if (name == "Name") {
    System.out.println("I love you, " + name + ", I have a happy family!");
  }

  **This example code will compile, but it will never print the message, even if the user does type "Barney".**
  ```

Another example

- The `==` operator on objects actually compares whether two variables refer to the same object.
- The `equals` method compares whether two objects have the same state as each other.

  **Given the following code:**
  ```java
  String p1 = new String("Hi");
  String p2 = new String("Hi");
  String p3 = p1;
  **What is printed?**
  if (p1 == p2) {
    System.out.println("1");
  } else if (p1 == p3) {
    System.out.println("2");
  } else {
    System.out.println("3");
  }
  ```

String condition methods

- There are several methods of a `String` object that can be used as conditions in `if` statements:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(String)</td>
<td>whether two strings contain exactly the same characters</td>
</tr>
<tr>
<td>equalsIgnoreCase(String)</td>
<td>whether two strings contain the same characters, ignoring uppercase/lowercase differences</td>
</tr>
<tr>
<td>startsWith(String)</td>
<td>whether one string contains the other's characters at its start</td>
</tr>
<tr>
<td>contains(String)</td>
<td>whether one string contains the other's characters at any position</td>
</tr>
</tbody>
</table>

String condition examples

- Hypothetical examples, assuming the existence of various `String` variables:
  ```java
  if (fullName.equals("John Doe");
  System.out.println("You are in the wrong town!");
  }
  if (fullName.startsWith("Jane");
  System.out.println("Your name is wrong!");
  }
  if (lastName.equals("Smith");
  System.out.println("You are a Smith family!");
  }
  if (name.toLowerCase().endsWith("xyz");
  System.out.println("You are a cool name.");
  }
  ```

Factoring if/else code

- **Factoring**: extracting common/redundant code
- Factoring if/else code reduces the size of the if/else statements and can sometimes eliminate the need for if/else altogether.

  ```java
  int x;
  if (a == 1) {
    x = 3;
  } else if (a == 2) {
    x = 5;
  } else if (a == 3) {
    x = 7;
  }
  ```
Code in need of factoring

* The following example has a lot of redundant code:

```java
if (money < 500) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Do you want to buy something?");
} else if (money < 1000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Think about it.");
} else if (money < 2000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Are you sure?");
} else if (money < 3000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Is that all? Do you want to buy something?");
} else {
    System.out.print("You have, ", money + " left.");
    System.out.print("You may bet literally.");
    System.out.print("How much do you want to bet?");
    bet = console.readInt();
}
```

Code after factoring

* Factoring tips:

  1. If the start of each branch is the same, move it below the if (line).
  2. If the end of each branch is the same, move it after the if (line).

```java
if (money < 500) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Do you want to buy something?");
} else if (money < 1000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Think about it.");
} else if (money < 2000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Are you sure?");
} else if (money < 3000) {
    System.out.print("You have, ", money + " left.");
    System.out.print("Is that all? Do you want to buy something?");
} else {
    System.out.print("You have, ", money + " left.");
    System.out.print("You may bet literally.");
    System.out.print("How much do you want to bet?");
    bet = console.readInt();
}
```

Type char

* char: A primitive type representing single characters.
* Individual characters inside a string are stored as char values.
* Literal char values are surrounded with apostrophes.
* (Single-quote) marks, such as 'a' or 'e' or '5' or 'Z'.

* It's legal to have variables, parameters, return of type char
  character letter - 'a',
  ```
  System.out.println(letter); // a
  ```

The charAt method

* The characters of a string can be accessed as char values using the string object's charAt method.

```java
String word = "hello!";
char firstLetter = word.charAt(0);
if (firstLetter == 'a') {
    System.out.println("That's a good enough for me!");
}
```

* We often use for loops that print or examine each character:

```java
String name = "Mari";
for (int i = 0; i < name.length(); i++) {
    System.out.println(name.charAt(i));
    Output:
    M
    a
    r
    i
    e
```

Text processing

* Text processing: Examining, editing, formatting text.
* Text processing often involves loops that examine the characters of a string one by one.
* You can use for loops to search for or count occurrences of a particular value in a string.

```java
// Returns the count of occurrences of a letter.
public static int countLetter(String str, char c) {
    int count = 0;
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == c) {
            count++;
        }
    }
    return count;
    count("Hello!", 'l') return 3
```
char Values can be concatenated with strings.

```java
char initial = 'F';
System.out.println(initial + " study");
```

You can compare char values with relational operators:
- `<` and `>`
- Note that you cannot use these operators on a String.

An example that prints the alphabet:
```java
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

char/int and type casting

- All char values are assigned numbers internally by the computer, called ASCII values.
- Examples:
  - 'a' is 97, 'z' is 122
  - Mixing char and int causes automatic conversion to int.
  - `a + 1` is 100, `a + 1` is 130
- To convert integer into the equivalent character, type cast it.
  - `char c = 'a' + 1;`

char vs. String

- 'b' is a char.
  ```java
  char c = 'b';
  ```
- char values are primitive; you cannot call methods on them
- `c.toUpperCase()`
- 'b' is a String
  ```java
  String s = "b";
  ```
- Strings are objects; they contain methods that can be called
- `s.length()`, `s.charAt(0)`
- `s.toUpperCase()`

- What is `s + 1`?
- What is `c + 1`?
- What is `s + a`?
- What is `c + a`?

Text processing questions

- Write a method named `pigLatinWord` that accepts a String as a parameter and outputs that word in simplified Pig Latin, by placing the word's first letter at the end followed by the suffix `ay`.
  ```java
  pigLatinWord("hello") prints el-hay
  pigLatinWord("goodbye") prints oodbye-gay
  ```
- Write methods named `encode` and `decode` that accept a String as a parameter and outputs that String with each of its letters increased or decreased by 1.
  ```java
  encode("hello") prints ifemp
  decode("ifemp") prints hello
  ```

Text processing question

- Write a method `printName` that accepts a full name as a parameter, and prints the last name followed by a comma, followed by the first name and middle initial.
  ```java
  printName("James Tiberius Kirk"); would output Kirk, James T.
  ```

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>charAt(index)</code></td>
<td>character at a specific index</td>
</tr>
<tr>
<td><code>indexOf(str)</code></td>
<td>index where the start of the given string appears in this string (if it is not there)</td>
</tr>
<tr>
<td><code>length()</code></td>
<td>number of characters in this string</td>
</tr>
<tr>
<td><code>substring(start, end)</code></td>
<td>the characters in this string between <code>start</code> (inclusive) to <code>end</code> (exclusive)</td>
</tr>
<tr>
<td><code>toLowerCase()</code></td>
<td>a new string with all uppercase letters lower-cased</td>
</tr>
</tbody>
</table>

Methods with if/else
if/else with return

- Methods can be written to return different values under different conditions using if/else statements:

```java
public static int min(int a, int b) {
    if (a < b) {
        return a;
    } else {
        return b;
    }
}
```

- Another example that returns the first word in a string:

```java
public static String firstWord(String s) {
    int index = s.indexOf(' ', 0); // Find the first space in the string
    if (index < 0) {
        return s; // Return the entire string if there is no space
    } else {
        return s.substring(0, index); // Return the substring before the space
    }
}
```

All code paths must return

- The following code also does not compile:

```java
public static int min(int a, int b) {
    if (a == b) {
        if (a < b) {
            return a;
        }
    }
}
```

- It produces the "Not all paths return a value" error.
  * To our eyes, it seems that all paths do return a value.
  * But the compiler thinks that it fails if code might choose not to
eexecute any branch, so it refuses to accept this code.

for loops with if/else return

- Methods with loops that return values must consider the case where the loop does not execute the return.

```java
public static int longestString(String[] strs) {
    for (int i = 0; i < strs.length; i++) {
        if (strs[i].length() == strs.length) {
            return strs[i];
        }
    }
    return -1; // Not found
}
```

- A better version that returns -1 when s is not found:

```java
public static int longestString(String[] strs) {
    for (int i = 0; i < strs.length; i++) {
        if (strs[i].length() == strs.length) {
            return strs[i];
        }
    }
    return -1; // Not found
}
```

if/else return question

- Write a method named `gcd` that accepts two integer parameters and returns their greatest common divisor.
  * For example, `gcd(9, 14)` returns 1 because 9 and 14 have no common factors other than 1.

- Write a method named `countFactors` that returns the number of factors an integer.
  * For example, `countFactors(6) returns 4 because 1, 2, 3, 4, 6, 12, 24, 36, and 48 are factors of 60.

- Modify the `piGcd` and `aCount` methods seen previously so that they return their results rather than printing them.

Method return question

- Write a program that prompts the user for a maximum integer and prints out a list of all prime numbers up to that maximum. Here is an example log of execution:

```
Maximum number: 50
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
14 total primes
```
Method return answer 1

```java
public class Primes {
    public static void main(String[] args) {
        int maxNumber = 100;
        System.out.println("Maximum number: "+maxNumber);
        System.out.println("Prime numbers up to "+maxNumber+":");
        for (int i = 2; i <= maxNumber; i++) {
            if (isPrime(i)) {
                System.out.println(i);
            }
        }
    }
    public static boolean isPrime(int number) {
        for (int i = 2; i <= number / 2; i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Method return answer 2

```java
public class Primes {
    public static void main(String[] args) {
        int maxNumber = 100;
        System.out.println("Maximum number: "+maxNumber);
        System.out.println("Prime numbers up to "+maxNumber+":");
        for (int i = 2; i <= maxNumber; i++) {
            if (isPrime(i)) {
                System.out.println(i);
            }
        }
    }
    public static boolean isPrime(int number) {
        for (int i = 2; i <= number / 2; i++) {
            if (number % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```