Building Java Programs, 3rd Edition
Exercise Solutions

NOTE: Answers to exercises are considered a private resource for instructors. Please do not post these answers on a public web site. Other instructors assign these problems as homework and do not want the answers to become publicly available. Thank you.

Many exercises can be solved in more than one way. Some exercises have more than one solution shown.

Chapter 1

1.

```
public class Stewie {
    public static void main(String[] args) {
        System.out.println("//////////////////////");
        System.out.println("|| Victory is mine! ||");
        System.out.println("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
    }
}
```

2.

```
public class Spikey {
    public static void main(String[] args) {
        System.out.println("  \\/");
        System.out.println(" \\\\//");
        System.out.println("\\\\\\///");
        System.out.println("///\\\\\\");
        System.out.println(" //\\\\");
        System.out.println("  /\\");
    }
}
```

3.

```
public class WellFormed {
    public static void main(String[] args) {
        System.out.println("A well-formed Java program has");
        System.out.println("a main method with { and }");
        System.out.println("braces.");
        System.out.println();
        System.out.println("A System.out.println statement");
        System.out.println("has ( and ) and usually a");
        System.out.println("String that starts and ends");
        System.out.println("with a \" character.");
        System.out.println("(But we type \\\" instead!)");
    }
}
```

4.

```
public class Difference {
    public static void main(String[] args) {
        System.out.println("What is the difference between");
        System.out.println("a ' and a \"?  Or between a \" and a \\\"?");
        System.out.println();
        System.out.println("One is what we see when we're typing our program.");
        System.out.println("The other is what appears on the \"console.\"");
    }
}
```

5.

```
public class MuchBetter {
    public static void main(String[] args) {
        System.out.println("A \"quoted\" String is");
        System.out.println("'much' better if you learn");
        System.out.println("the rules of \"escape sequences.\"");
        System.out.println("Also, \"\" represents an empty String.");
        System.out.println("Don't forget: use \\\" instead of \" !");
        System.out.println("'' is not the same as \"");
    }
}
```

6.

```
public class Meta {
    public static void main(String[] args) {
        System.out.println("public class Hello {");
        System.out.println("    public static void main(String[] args) {");
        System.out.println("        System.out.println(\"Hello, world!\");");
        System.out.println("    }");
        System.out.println("}");
    }
}
```

7.

```
public class Mantra {
    public static void main(String[] args) {
        message();
        System.out.println();
        message();
    }
    public static void message() {
        System.out.println("There's one thing every coder must understand:");
        System.out.println("The System.out.println command.");
    }
}
```

8.

```
// This program prints a message multiple times using static methods.
public class Stewie2 {
    public static void main(String[] args) {
        System.out.println("//////////////////////");
        printVictory();
        printVictory();
        printVictory();
        printVictory();
        printVictory();
    }
    public static void printVictory() {
        System.out.println("|| Victory is mine! ||");
        System.out.println("\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\");
    }
}
```

9.

```
// Draws an egg figure.
public class Egg {
    public static void main(String[] args) {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("-\"-'-\"-'-\"-");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
    }
}
```

10.

```
// Draws several egg figures.
public class Egg2 {
    public static void main(String[] args) {
        drawEgg();
        drawEgg();
        drawBottom();
        drawTop();
        drawLine();
        drawBottom();
    }
    public static void drawEgg() {
        drawTop();
        drawBottom();
        drawLine();
    }
    public static void drawTop() {
        System.out.println("  _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }
    public static void drawBottom() {
        System.out.println("\\         /");
        System.out.println(" \\_______/");
    }
    public static void drawLine() {
        System.out.println("-\"-'-\"-'-\"-");
    }
}
```

11.

```
// Draws two rocket ship figures side-by-side.
public class TwoRockets {
    public static void main(String[] args) {
        printTop();
        printSquare();
        printLabel();
        printSquare();
        printTop();
    }
    public static void printTop() {
        System.out.println("   /\\       /\\");
        System.out.println("  /  \\     /  \\");
        System.out.println(" /    \\   /    \\");
    }
    public static void printSquare() {
        System.out.println("+------+ +------+");
        System.out.println("|      | |      |");
        System.out.println("|      | |      |");
        System.out.println("+------+ +------+");
    }
    public static void printLabel() {
        System.out.println("|United| |United|");
        System.out.println("|States| |States|");
    }
}
```

12.

```java
// This program prints a college "fight song" with verses and repetition.
// The code uses static methods for structure and to remove redundancy.
public class FightSong {
    public static void main(String[] args) {
        goTeamGo();
        System.out.println();
        bigVerse();
        bigVerse();
        goTeamGo();
    }

    public static void goTeamGo() {
        System.out.println("Go, team, go!");
        System.out.println("You can do it.");
    }

    public static void bigVerse() {
        goTeamGo();
        System.out.println("You're the best,");
        System.out.println("In the West.");
        goTeamGo();
        System.out.println();
    }
}
```

13.

```java
// This program prints a pattern of starry figures.
public class StarFigures {
    public static void main(String[] args) {
        printFigure1();
        System.out.println();
        printFigure2();
        System.out.println();
        printFigure3();
    }
    public static void printFigure1() {
        printHorizontalBar();
        printX();
    }
    public static void printFigure2() {
        printHorizontalBar();
        printX();
        printHorizontalBar();
    }
    public static void printFigure3() {
        System.out.println("  *");
        System.out.println("  *");
        System.out.println("  *");
        printFigure1();
    }
    public static void printHorizontalBar() {
        System.out.println("*****");
        System.out.println("*****");
    }
    public static void printX() {
        System.out.println(" * *");
        System.out.println("  *");
        System.out.println(" * *");
    }
}
```

14.

```java
// This program prints a pattern of vaguely lantern-like figures.
// The code uses static methods for structure and to remove redundancy.
public class Lanterns {
    public static void main(String[] args) {
        figure1();
        System.out.println();

        lantern1();
        System.out.println();

        lantern2();
        System.out.println();
    }

    public static void figure1() {
        shortLong();
    }

    public static void lantern1() {
        shortLong();
        middle();
        longLine();
        shortLong();
    }

    public static void lantern2() {
        shortLong();
        shortLine();
        middle();
        middle();
        shortLine();
        shortLine();
    }

    public static void shortLong() {
        shortLine();
        System.out.println("  *********");
        longLine();
    }

    public static void shortLine() {
        System.out.println("    *****");
    }

    public static void longLine() {
        System.out.println("*************");
    }

    public static void middle() {
        System.out.println("* | | | | | *");
    }
}
```

15.

```java
// This program prints a pattern of figures such as eggs and stop signs.
// The code uses static methods for structure and to remove redundancy.
public class EggStop {
    public static void main(String[] args) {
        egg();
        System.out.println();
        egg();
        line();
        System.out.println();
        stopSign();
        line();
        System.out.println();
    }
    public static void egg() {
        eggTop();
        eggBottom();
    }
    public static void eggTop() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
    }
    public static void eggBottom() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
    }
    public static void stopSign() {
        eggTop();
        System.out.println("|  STOP  |");
        eggBottom();
    }
    public static void line() {
        System.out.println("+--------+");
    }
}
```

16.

```
// This program prints a memorable movie quote 1000 times.
public class Shining {
    public static void main(String[] args) {
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
        allWork3();
    }
    public static void allWork3() {
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
        allWork2();
    }
    public static void allWork2() {
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
        allWork1();
    }
    public static void allWork1() {
        System.out.println("All work and no play makes Jack a dull boy.");
    }
}
```

Chapter 2

1.

```
double s0 = 12.0;
double v0 = 3.5;
double a = 9.8;
int t = 10;
double s = s0 + v0 * t + a * t * t / 2.0;
System.out.println(s);
```

2.

```
int number = 1;
int increment = 3;
for (int i = 1; i <= 10; i++) {
    System.out.print(number + " ");
    number = number + increment;
    increment = increment + 2;
}
System.out.println();   // to end the line
```

```
for (int i = 1; i <= 10; i++) {
    System.out.print(i * i + " ");
}
System.out.println();   // to end the line
```

3.

```
int n1 = 1;
int n2 = 1;
System.out.print(n1 + " " + n2 + " ");
for (int i = 3; i <= 12; i++) {
    int n3 = n1 + n2;
    n1 = n2;
    n2 = n3;
    System.out.print(n2 + " ");
}
System.out.println();
```

4.

```
for (int i = 1; i <= 4; i++) {
    for (int j = 1; j <= 5; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

5.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

6.

```
for (int i = 1; i <= 7; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

7.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5 - i; j++) {
        System.out.print(" ");
    }
    System.out.println(i);
}
```

8.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 5 - i; j++) {
        System.out.print(" ");
    }
    for (int nums = 1; nums <= i; nums++) {
        System.out.print(i);
    }
    System.out.println();
}
```

9.

```
int count = 20;
for (int i = 1; i <= count; i++) {
    System.out.print("--");
}
System.out.println();
for (int i = 0; i < count / 2; i++) {
    System.out.print("_-^-");
}
System.out.println();
for (int i = 1; i <= count; i++) {
    System.out.print(i % 10);
    System.out.print(i % 10);
}
System.out.println();
for (int i = 1; i <= count; i++) {
    System.out.print("--");
}
System.out.println();
```

10.

```
for (int i = 1; i <= 6; i++) {
    System.out.print("         |");
}
System.out.println();
for (int i = 1; i <= 6; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print(j % 10);
    }
}
System.out.println();
```

11.

```
public class NumberOutput2 {
    public static final int COUNT = 6;
    public static final int INNER_COUNT = 10;
    public static void main(String[] args) {
        for (int i = 1; i <= COUNT; i++) {
            for (int j = 1; j <= INNER_COUNT - 1; j++) {
                System.out.print(" ");
            }
            System.out.print("|");
        }
        System.out.println();
        for (int i = 1; i <= COUNT; i++) {
            for (int j = 1; j <= INNER_COUNT; j++) {
                System.out.print(j % INNER_COUNT);
            }
        }
        System.out.println();
    }
}
```

12.

```
for (int i = 1; i <= 3; i++) {
    for (int j = 0; j <= 9; j++) {
        for (int k = 1; k <= 3; k++) {
            System.out.print(j);
        }
    }
    System.out.println();
}
```

13.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 9; j >= 0; j--) {
        for (int k = 1; k <= 5; k++) {
            System.out.print(j);
        }
    }
    System.out.println();
}
```

14.

```
for (int i = 1; i <= 4; i++) {
    for (int j = 9; j >= 0; j--) {
        for (int k = 1; k <= j; k++) {
            System.out.print(j);
        }
    }
    System.out.println();
}
```

15.

```
public static void printDesign() {
    for (int line = 1; line <= 5; line++) {
        for (int dash = 1; dash <= -1 * line + 6; dash++) {
            System.out.print("-");
        }
        for (int number = 1; number <= 2 * line - 1; number++) {
            System.out.print(2 * line - 1);
        }
        for (int dash = 1; dash <= -1 * line + 6; dash++) {
            System.out.print("-");
        }
        System.out.println();
    }
}
```

16.

```
public class SlashFigure {
    public static void main(String[] args) {
        for (int line = 1; line <= 6; line++) {
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("\\");
            }
            for (int i = 1; i <= -4 * line + 26; i++) {
                System.out.print("!");
            }
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("/");
            }
            System.out.println();
        }
    }
}
```

17.

```
public class SlashFigure2 {
    public static final int SIZE = 4;
    public static void main(String[] args) {
        for (int line = 1; line <= SIZE; line++) {
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("\\");
            }
            for (int i = 1; i <= -4 * line + (4 * SIZE + 2); i++) {
                System.out.print("!");
            }
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("/");
            }
            System.out.println();
        }
    }
}
```

18.

```
overall algorithm:
    draw a horizontal line
    draw 3 lines of bars
    draw a line
    draw 3 lines of bars
    draw a line
how to draw a horizontal line:
    print a +
    print 3 = signs
    System.out.print("+");
    print a +
    print 3 = signs
    print a +
how to draw a line of bars:
    print a |
    print 3 spaces
    print a |
    print 3 spaces
    print a |
```

19.

```java
// Draws a resizable window figure with nested for loops
// and a class constant.
public class Window {
    public static final int COUNT = 3;
    public static void main(String[] args) {
        drawLine();
        for (int i = 1; i <= 2; i++) {
            for (int j = 1; j <= COUNT; j++) {
                drawBars();
            }
            drawLine();
        }
    }
    // Draws a horizontal line: +===+===+
    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= COUNT; i++) {
            System.out.print("=");
        }
        System.out.print("+");
        for (int i = 1; i <= COUNT; i++) {
            System.out.print("=");
        }
        System.out.println("+");
    }
    // Draws a single line of bars: |   |   |
    public static void drawBars() {
        System.out.print("|");
        for (int i = 1; i <= COUNT; i++) {
            System.out.print(" ");
        }
        System.out.print("|");
        for (int i = 1; i <= COUNT; i++) {
            System.out.print(" ");
        }
        System.out.println("|");
    }
}
```

20.

```java
public class StarFigure {
    public static void main(String[] args) {
        for (int line = 1; line <= 5; line++) {
            for (int i = 1; i <= -4 * line + 20; i++) {
                System.out.print("/");
            }
            for (int i = 1; i <= 8 * line - 8; i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -4 * line + 20; i++) {
                System.out.print("\\");
            }
            System.out.println();
        }
    }
}
```

21.

```
public class StarFigure2 {
    public static final int SIZE = 5;

    public static void main(String[] args) {
        for (int line = 1; line <= SIZE; line++) {
            for (int i = 1; i <= -4 * line + 4 * SIZE; i++) {
                System.out.print("/");
            }
            for (int i = 1; i <= 8 * line - 8; i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -4 * line + 4 * SIZE; i++) {
                System.out.print("\\");
            }
            System.out.println();
        }
    }
}
```

22.

```
public class DollarFigure {
    public static void main(String[] args) {
        for (int line = 1; line <= 7; line++) {
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -1 * line + 8; i++) {
                System.out.print("$");
            }
            for (int i = 1; i <= -2 * line + 16; i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -1 * line + 8; i++) {
                System.out.print("$");
            }
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

23.

```
public class DollarFigure2 {
    public static final int SIZE = 7;

    public static void main(String[] args) {
        for (int line = 1; line <= SIZE; line++) {
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -1 * line + (SIZE + 1); i++) {
                System.out.print("$");
            }
            for (int i = 1; i <= -2 * line + (2 * SIZE + 2); i++) {
                System.out.print("*");
            }
            for (int i = 1; i <= -1 * line + (SIZE + 1); i++) {
                System.out.print("$");
            }
            for (int i = 1; i <= 2 * line - 2; i++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Chapter 3

1.

```
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
}
```

2.

```
public static void printPowersOf2(int max) {
    for (int i = 0; i <= max; i++) {
        System.out.print((int) Math.pow(2, i) + " ");
    }
    System.out.println();  // to end the line of output
}
```

```
public static void printPowersOf2(int max) {
    int power = 1;
    for (int i = 0; i <= max; i++) {
        System.out.print(power + " ");
        power = power + power;
    }
    System.out.println();  // to end the line of output
}
```

3.

```
public static void printPowersOfN(int base, int exp) {
    for (int i = 0; i <= exp; i++) {
        System.out.print((int) Math.pow(base, i) + " ");
    }
    System.out.println();  // to end the line of output
}
```

```
public static void printPowersOfN(int base, int exp) {
    int power = 1;
    for (int i = 0; i <= exp; i++) {
        System.out.print(power + " ");
        power = power * base;
    }
    System.out.println();  // to end the line of output
}
```

4.

```
public static void printSquare(int min, int max) {
    int range = max - min + 1;
    for (int i = 0; i < range; i++) {
        for (int j = 0; j < range; j++) {
            System.out.print((j + i) % range + min);
        }
        System.out.println();
    }
}
```

```
public static void printSquare(int min, int max) {
    int range = max - min + 1;
    for (int i = 0; i < range; i++) {
        for (int j = min + i; j <= max; j++) {
            System.out.print(j);
        }
        for (int j = min; j < min + i; j++) {
            System.out.print(j);
        }
        System.out.println();
    }
}
```

5.

```
public static void printGrid(int rows, int cols) {
    for (int i = 1; i <= rows; i++) {
        System.out.print(i);
        for (int j = 1; j <= cols - 1; j++) {
            System.out.print(", " + (i + rows * j));
        }
        System.out.println();
    }
}
```

```
public static void printGrid(int rows, int cols) {
    for (int i = 1; i <= rows; i++) {
        for (int j = 0; j < cols - 1; j++) {
            System.out.print((i + rows * j) + ", ");
        }
        System.out.println(i + rows * (cols - 1));
    }
}
```

```
public static void printGrid(int rows, int cols) {
    int n = 1;
    int count1 = 1;
    int count2 = 1;
    while (count1 <= rows * cols) {
        if (count1 % cols == 0) {
            System.out.println(n);
            count2++;
            n = count2;
        } else {
            System.out.print(n + ", ");
            n = n + rows;
        }
        count1++;
    }
}
```

6.

```
public static int largerAbsVal(int n1, int n2) {
    return Math.max(Math.abs(n1), Math.abs(n2));
}
```

7.

```
public static int largestAbsVal(int n1, int n2, int n3) {
    int larger12 = Math.max(Math.abs(n1), Math.abs(n2));
    int larger23 = Math.max(Math.abs(n2), Math.abs(n3));
    return Math.max(larger12, larger23);
}
```

```
public static int largestAbsVal(int n1, int n2, int n3) {
    return Math.max(largerAbsVal(n1, n2), largerAbsVal(n2, n3));
}
public static int largerAbsVal(int n1, int n2) {
    return Math.max(Math.abs(n1), Math.abs(n2));
}
```

8.

```
public static void quadratic(int a, int b, int c) {
    double determinant = b * b - 4 * a * c;
    double root1 = (-b + Math.sqrt(determinant)) / (2 * a);
    double root2 = (-b - Math.sqrt(determinant)) / (2 * a);
    System.out.println("First root = " + root1);
    System.out.println("Second root = " + root2);
}
```

9.

```
public static int lastDigit(int num) {
    return Math.abs(num) % 10;
}
```

10.

```
public static double area(double radius) {
    double answer = Math.PI * Math.pow(radius, 2);
    return answer;
}
```

11.

```
public static double distance(int x1, int y1, int x2, int y2) {
    int dx = x2 - x1;
    int dy = y2 - y1;
    return Math.sqrt(Math.pow(dx, 2) + Math.pow(dy, 2));
}
```

```
public static double distance(int x1, int y1, int x2, int y2) {
    int dx = x2 - x1, dy = y2 - y1;
    return Math.sqrt(dx * dx + dy * dy);
}
```

12.

```
public static double scientific(double base, double exponent) {
    return base * Math.pow(10, exponent);
}
```

13.

```java
public static double pay(double salary, int hours) {
    if (hours < 8) {
        return salary * hours;
    } else {
        double regularPay = salary * 8;
        double overtimePay = (1.5 * salary) * (hours - 8);
        return regularPay + overtimePay;
    }
}
```

```java
public static double pay(double salary, int hours) {
    return salary * (hours + 0.5 * Math.max(0, hours - 8));
}
```

14.

```java
public static double cylinderSurfaceArea(double r, double h) {
    return 2 * Math.PI * r * r + 2 * Math.PI * r * h;
}
```

15.

```java
public static double sphereVolume(double r) {
    return 4.0 * Math.PI * r * r * r / 3.0;
}
```

16.

```java
public static double triangleArea(double a, double b, double c) {
    double s = (a + b + c) / 2.0;
    return Math.sqrt(s * (s - a) * (s - b) * (s - c));
}
```

17.

```java
public static String padString(String s, int length) {
    String spaces = "";
    for (int i = 0; i < length - s.length(); i++) {
        spaces += " ";
    }
    return spaces + s;
}
```

18.

```java
public static void vertical(String str) {
    for (int i = 0; i < str.length(); i++) {
        System.out.println(str.charAt(i));
    }
}
```

19.

```java
public static void printReverse(String str) {
    for (int i = str.length() - 1; i >= 0; i--) {
        System.out.print(str.charAt(i));
    }
    System.out.println();
}
```

20.

```
public static void inputBirthday(Scanner input) {
    System.out.print("On what day of the month were you born? ");
    int day = input.nextInt();
    System.out.print("What is the name of the month in which you were born? ");
    String month = input.next();
    System.out.print("During what year were you born? ");
    String year = input.next();
    System.out.print("You were born on " + month + " " + day + ", "
                     + year + ". You're mighty old!");
}
```

21.

```
public static void processName(Scanner input) {
    System.out.print("Please enter your full name: ");
    String first = input.next();
    String last = input.next();
    System.out.println();
    System.out.print("Your name in reverse order is " + last + ", " + first);
}
```

22.

```
public class CollegeAdmit {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.println("University admission program");

        System.out.print("What is your GPA? ");
        double gpa = console.nextDouble();
        System.out.print("What is your SAT score? ");
        int sat = console.nextInt();

        if (gpa < 1.8) {
            System.out.println("Your GPA is too low.");
        } else if (sat < 900) {
            System.out.println("Your SAT score is too low.");
        } else {
            System.out.println("You were accepted!");
        }
    }
}
```

Supplement 3G

1.

```
public class MickeyBox {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(220, 150);
        panel.setBackground(Color.YELLOW);
        Graphics g = panel.getGraphics();
        g.setColor(Color.BLUE);
        g.fillOval(50, 25, 40, 40);
        g.fillOval(130, 25, 40, 40);
        g.setColor(Color.RED);
        g.fillRect(70, 45, 80, 80);
        g.setColor(Color.BLACK);
        g.drawLine(70, 85, 150, 85);
    }
}
```

2.

```
public class MickeyBox2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(450, 150);
        panel.setBackground(Color.YELLOW);
        Graphics g = panel.getGraphics();
        drawFigure(g, new Point(50, 25));
        drawFigure(g, new Point(250, 45));
    }
    public static void drawFigure(Graphics g, Point location) {
        g.setColor(Color.BLUE);
        g.fillOval(location.x, location.y, 40, 40);
        g.fillOval(location.x + 80, location.y, 40, 40);
        g.setColor(Color.RED);
        g.fillRect(location.x + 20, location.y + 20, 80, 80);
        g.setColor(Color.BLACK);
        g.drawLine(location.x + 20, location.y + 60, location.x + 100, location.y + 60);
    }
}
```

3.

```
public class Face {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(320, 180);
        Graphics g = panel.getGraphics();
        drawFace(g, 10, 30);
        drawFace(g, 150, 50);
    }

    public static void drawFace(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.drawOval(x, y, 100, 100);
        g.setColor(Color.BLUE);
        g.fillOval(x + 20, y + 30, 20, 20);
        g.fillOval(x + 60, y + 30, 20, 20);
        g.setColor(Color.RED);
        g.drawLine(x + 30, y + 70, x + 70, y + 70);
    }
}
```

4.

```
public class Face2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(520, 180);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 5; i++) {
            drawFace(g, 10 + i * 100, 30);
        }
    }

    public static void drawFace(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.drawOval(x, y, 100, 100);
        g.setColor(Color.BLUE);
        g.fillOval(x + 20, y + 30, 20, 20);
        g.fillOval(x + 60, y + 30, 20, 20);
        g.setColor(Color.RED);
        g.drawLine(x + 30, y + 70, x + 70, y + 70);
    }
}
```

5.

```
public class ShowDesign {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 200);
        panel.setBackground(Color.WHITE);
        Graphics g = panel.getGraphics();
        g.setColor(Color.BLACK);
        for (int i = 1; i <= 4; i++) {
            int x = i * 20;
            int y = i * 20;
            int w = (10 - 2 * i) * 20;
            int h = (10 - 2 * i) * 20;
            g.drawRect(x, y, w, h);
        }
    }
}
```

6.

```
public class ShowDesign2 {
    public static void main(String[] args) {
        showDesign(300, 100);
    }
    public static void showDesign(int width, int height) {
        DrawingPanel panel = new DrawingPanel(width, height);
        panel.setBackground(Color.WHITE);
        Graphics g = panel.getGraphics();
        g.setColor(Color.BLACK);
        for (int i = 1; i <= 4; i++) {
            int x = i * width  / 10;
            int y = i * height / 10;
            int w = (10 - 2 * i) * width / 10;
            int h = (10 - 2 * i) * height / 10;
            g.drawRect(x, y, w, h);
        }
    }
}
```

7.

```
public class Squares {
    public static void main(String[] args) {
        DrawingPanel p = new DrawingPanel(300, 200);
        p.setBackground(Color.CYAN);
        Graphics g = p.getGraphics();
        g.setColor(Color.RED);
        for (int i = 1; i <= 5; i++) {
            g.drawRect(50, 50, i * 20, i * 20);
        }
        g.setColor(Color.BLACK);
        g.drawLine(50, 50, 150, 150);
    }
}
```

8.

```
public class Squares2 {
    public static void main(String[] args) {
        DrawingPanel p = new DrawingPanel(400, 300);
        p.setBackground(Color.CYAN);
        Graphics g = p.getGraphics();
        drawFigure(g, 50, 50);
        drawFigure(g, 250, 10);
        drawFigure(g, 180, 115);
    }
    public static void drawFigure(Graphics g, int x, int y) {
        g.setColor(Color.RED);
        for (int i = 1; i <= 5; i++) {
            g.drawRect(x, y, i * 20, i * 20);
        }
        g.setColor(Color.BLACK);
        g.drawLine(x, y, x + 100, y + 100);
    }
}
```

9.

```
public class Squares3 {
    public static void main(String[] args) {
        DrawingPanel p = new DrawingPanel(400, 300);
        p.setBackground(Color.CYAN);
        Graphics g = p.getGraphics();
        drawFigure(g, 50, 50, 100);
        drawFigure(g, 250, 10, 50);
        drawFigure(g, 180, 115, 180);
    }
    public static void drawFigure(Graphics g, int x, int y, int size) {
        g.setColor(Color.RED);
        for (int i = 1; i <= 5; i++) {
            g.drawRect(x, y, i * size / 5, i * size / 5);
        }
        g.setColor(Color.BLACK);
        g.drawLine(x, y, x + size, y + size);
    }
}
```

10.

```
public class Stairs {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(110, 110);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 10; i++) {
            g.drawRect(5, 5 + 10 * i, 10 + 10 * i, 10);
        }
    }
}
```

11.

```
public class Stairs2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(110, 110);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 10; i++) {
            g.drawRect(5, 5 + 10 * i, 100 - 10 * i, 10);
        }
    }
}

public class Stairs3 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(110, 110);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 10; i++) {
            g.drawRect(95 - 10*i, 5 + 10*i, 10 + 10*i, 10);
        }
    }
}

public class Stairs4 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(110, 110);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 10; i++) {
            g.drawRect(5 + 10*i, 5 + 10*i, 100 - 10*i, 10);
        }
    }
}
```

12.

```
public class Triangle {
    public static void main(String[] args) {
        DrawingPanel p = new DrawingPanel(600, 200);
        p.setBackground(Color.YELLOW);
        Graphics g = p.getGraphics();
        g.setColor(Color.BLUE);
        g.drawLine(0, 0, 300, 200);
        g.drawLine(300, 200, 600, 0);
        for (int i = 1; i <= 19; i++) {
            g.drawLine(15 * i, 10 * i, 600 - 15 * i, 10 * i);
        }
    }
}
```

Chapter 4

1.

```
public static double fractionSum(int n) {
    int denominator = 1;
    double sum = 0.0;
    for (int i = 1; i <= n; i++) {
        sum += (double) 1 / denominator;
        denominator++;
    }
    return sum;
}
```

2.

```
// Returns s concatenated together n times.
// If n <= 0, an empty string is returned.
public static String repl(String s, int n) {
    String result = "";
    for (int i = 0; i < n; i++) {
        result = result + s;
    }
    return result;
}
```

3.

```
public static String season(int month, int day) {
    if (month < 3 || (month == 3 && day < 16)) {
        return "Winter";
    } else if (month < 6 || (month == 6 && day < 16)) {
        return "Spring";
    } else if (month < 9 || (month == 9 && day < 16)) {
        return "Summer";
    } else if (month < 12 || (month == 12 && day < 16)) {
        return "Fall";
    } else { // last part of the year in December
        return "Winter";
    }
}
```

```
public static String season(int month, int day) {
    if (month < 3 || (month == 3 && day < 16) || (month == 12 && day >= 16)) {
        return "Winter";
    } else if (month < 6 || (month == 6 && day < 16)) {
        return "Spring";
    } else if (month < 9 || (month == 9 && day < 16)) {
        return "Summer";
    } else {
        return "Fall";
    }
}
```

4.

```
public static int daysInMonth(int month) {
    if (month == 4 || month == 6 || month == 9 || month == 11) {
        return 30;          // 30 days have September, April, June, and November
    } else if (month == 2) {
        return 28;          // 28 days in February in a non-leap year
    } else {
        return 31;          // 31 days in all other months
    }
}
```

5.

```
// Returns base raised to exponent power.
// Preconditions: base and exponent >= 0
public static int pow(int base, int exponent) {
    if (base == 0) {
        // 0 to any power is 0
        return 0;
    } else {
        // multiply the base together, exponent times (cumulative product)
        int result = 1;
        for (int i = 1; i <= exponent; i++) {
            result *= base;
        }
        return result;
    }
}
```

6.

```
public static void printRange(int n1, int n2) {
    System.out.print("[");
    if (n1 <= n2) {
        for (int i = n1; i < n2; i++) {
            System.out.print(i + ", ");
        }
        System.out.println(n2+"]");
    } else {
        for (int i = n1; i > n2; i--) {
            System.out.print(i + ", ");
        }
        System.out.println(n2+"]");
    }
}
```

7.

```
public static void xo(int size) {
    for (int i = 1; i <= size; i++) {
        for (int j = 1; j <= size; j++) {
            if (j == i || j == size - (i - 1)) {   // note the (i - 1), not i
                System.out.print("x");
            } else {
                System.out.print("o");
            }
        }
        System.out.println();
    }
}
```

8.

```
public static void smallestLargest() {
    System.out.print("How many numbers do you want to enter? ");
    Scanner console = new Scanner(System.in);
    int count = console.nextInt();
    // read/print first number (fencepost solution)
    System.out.print("Number 1: ");
    int smallest = console.nextInt();
    int largest = smallest;
    // read/print remaining numbers
    for (int i = 2; i <= count; i++) {
        System.out.print("Number " + i + ": ");
        int num = console.nextInt();
        if (num > largest) {
            largest = num;
        } else if (num < smallest) {
            smallest = num;
        }
    }
    // print overall stats
    System.out.println("Smallest = " + smallest);
    System.out.println("Largest = " + largest);
}
```

9.

```
public static void evenSumMax() {
    Scanner console = new Scanner(System.in);
    System.out.print("how many integers? ");
    int count = console.nextInt();
    int sum = 0;
    int max = 0;
    for (int i = 1; i <= count; i++) {
        System.out.print("next integer? ");
        int next = console.nextInt();
        if (next % 2 == 0) {
            sum = sum + next;
            if (next > max) {
                max = next;       // or,  max = Math.max(max, next);
            }
        }
    }
    System.out.println("even sum = " + sum);
    System.out.println("even max = " + max);
}
```

10.

```
public static void printGPA() {
    System.out.print("Enter a student record: ");
    Scanner console = new Scanner(System.in);
    String name = console.next();
    int count = console.nextInt();
    int sum = 0;
    for (int i = 1; i <= count; i++) {
        sum = sum + console.nextInt();
    }
    double average = (double) sum / count;
    System.out.println(name + "'s grade is " + average);
}
```

11.

```
public static void longestName(Scanner console, int names) {
    System.out.print("name #1? ");
    String longest = console.next();
    int count = 1;
    for (int i = 2; i <= names; i++) {
        System.out.print("name #" + i + "? ");
        String name = console.next();
        if (name.length() == longest.length()) {
            count++;
        }
        if (name.length() > longest.length()) {
            longest = name;
            count = 1;
        }
    }
    String fixedName = longest.substring(0, 1).toUpperCase();
    fixedName = fixedName + longest.substring(1).toLowerCase();
    System.out.println(fixedName + "'s name is longest");
    if (count > 1) {
        System.out.println("(There was a tie!)");
    }
}
```

12.

```
public static void printTriangleType(int s1, int s2, int s3) {
    if (s1 == s2) {
        if (s2 == s3) {
            System.out.println("equilateral");
        }
    } else if (s1 == s2 || s1 == s3 || s2 == s3) {
        System.out.println("isosceles");
    } else {
        System.out.println("scalene");
    }
}
```

13.

```
public static double average(int n1, int n2) {
    return (n1 + n2) / 2.0;
}
```

14.

```
public static double pow2(double base, int exponent) {
    if (base == 0.0) {
        return 0;
    } else {
        double result = 1.0;
        for (int i = 1; i <= Math.abs(exponent); i++) {
            result *= base;
        }
        if (exponent < 0) {
            result = 1.0 / result;
        }
        return result;
    }
}
```

15.

```
public static double getGrade(int score) {
    if (score < 0 || score > 100) {
        throw new IllegalArgumentException();
    } else if (score >= 95) {
        return 4.0;
    } else if (score < 60) {
        return 0.0;
    } else if (score <= 62) {
        return 0.7;
    } else {
        return 0.1 * (score - 55);
    }
}
```

16.

```
public static void printPalindrome(Scanner console) {
    System.out.print("Type one or more words: ");
    String input = console.nextLine();
    String lcInput = input.toLowerCase();
    int matches = 0;
    for (int i = 0; i < lcInput.length() / 2; i++) {
        if (lcInput.charAt(i) == lcInput.charAt(lcInput.length() - 1 - i)) {
            matches++;
        }
    }
    if (matches == lcInput.length() / 2) {
        System.out.println(input + " is a palindrome!");
    } else {
        System.out.println(input + " is not a palindrome.");
    }
}
```

17.

```
public static String swapPairs(String s) {
    String result = "";
    for(int i = 0; i < s.length() - 1; i+= 2) {
        result += s.charAt(i + 1);
        result += s.charAt(i);
    }
    if(s.length() % 2 != 0) {
        result += s.charAt(s.length() - 1);
    }
    return result;
}
```

18.

```
public static int wordCount(String s) {
    int count = 0;
    if (s.charAt(0) != ' ') {
        count++;
    }
    for (int i = 0; i < s.length() - 1; i++) {
        if (s.charAt(i) == ' ' && s.charAt(i + 1) != ' ') {
            count++;
        }
    }
    return count;
}
```

19.

```
public static int quadrant(double x, double y) {
    if (x > 0 && y > 0) {
        return 1;
    } else if (x < 0 && y > 0) {
        return 2;
    } else if (x < 0 && y < 0) {
        return 3;
    } else if (x > 0 && y < 0) {
        return 4;
    } else {  // at least one equals 0
        return 0;
    }
}
```

20.

```
public static int numUnique(int a, int b, int c) {
    if (a == b && b == c) {
        return 1;
    } else if (a != b && b != c && a != c) {
        return 3;
    } else {
        return 2;
    }
}
```

21.

```
public static void perfectNumbers(int n) {
    System.out.print("Perfect numbers up to " + n + ":");
    for (int current = 1; current <= n; current++) {
        int sum = 0;
        for (int factor = 1; factor < current; factor++) {
            if (current % factor == 0) {
                sum = sum + factor;
            }
        }
        if (sum == current) {
            System.out.print(" " + current);
        }
    }
    System.out.println();
}
```

Chapter 5

1.

```
public static void showTwos(int n) {
    System.out.print(n + " = ");
    while (n % 2 == 0) {
        System.out.print("2 * ");
        n = n / 2;
    }
    System.out.println(n);
}
```

2.

```
public static int gcd(int a, int b) {
    while (b != 0) {
        int temp = a % b;
        a = b;
        b = temp;
    }
    return Math.abs(a);
}
```

3.

```
public static String toBinary(int number) {
    String binary = "";
    if (number == 0) {
        binary = "0";
    } else {
        while (number != 0) {
            binary = number % 2 + binary;
            number = number / 2;
        }
    }
    return binary;
}
```

4.

```
public static void randomX() {
    Random rand = new Random();
    int xCount = 0;   // dummy value; anything below 16
    while (xCount < 16) {
        xCount = rand.nextInt(15) + 5;   // random number from 5-19
        for (int j = 1; j <= xCount; j++) {
            System.out.print("x");
        }
        System.out.println();
    }
}
```

5.

```
public static void randomLines() {
    Random rand = new Random();
    int lines = rand.nextInt(6) + 5;
    for (int i = 1; i <= lines; i++) {
        int length = rand.nextInt(81);
        for (int j = 1; j <= length; j++) {
            char letter = (char) (rand.nextInt(26) + 'a');
            System.out.print(letter);
        }
        System.out.println();
    }
}
```

6.

```
public static void makeGuesses() {
    Random rand = new Random();
    int guess = 0;
    int count = 0;
    while (guess < 48) {
        count++;
        guess = rand.nextInt(50) + 1;
        System.out.println("guess = " + guess);
    }
    System.out.println("total guesses = " + count);
}
```

7.

```
public static void diceSum() {
    Scanner console = new Scanner(System.in);
    System.out.print("Desired dice sum: ");
    int desiredSum = console.nextInt();
    int die1 = 0;
    int die2 = 0;
    Random r = new Random();
    while (die1 + die2 != desiredSum) {
        die1 = r.nextInt(6) + 1;
        die2 = r.nextInt(6) + 1;
        int sum = die1 + die2;
        System.out.println(die1 + " and " + die2 + " = " + (die1 + die2));
    }
}
```

```
public static void diceSum() {
    Scanner console = new Scanner(System.in);
    System.out.print("Desired dice sum: ");
    int desiredSum = console.nextInt();
    int sum;
    Random r = new Random();
    do {
        int die1 = r.nextInt(6) + 1;
        int die2 = r.nextInt(6) + 1;
        sum = die1 + die2;
        System.out.println(die1 + " and " + die2 + " = " + sum);
    } while (sum != desiredSum);
}
```

8.

```
public static void randomWalk() {
    int n = 0;
    int max = 0;
    Random r = new Random();
    System.out.println("position = " + n);
    while (-3 < n && n < 3) {
        int flip = r.nextInt(2);
        if (flip == 0) {
            n++;
        } else {
            n--;
        }
        max = Math.max(n, max);
        System.out.println("position = " + n);
    }
    System.out.println("max position = " + max);
}
```

9.

```java
// Prints the factors of an integer separated by " and ".
// Precondition: n >= 1
public static void printFactors(int n) {
    // print first factor, always 1 (fencepost)
    System.out.print(1);
    // print remaining factors, if any, preceded by " and "
    for (int i = 2; i <= n; i++) {
        if (n % i == 0) {
            System.out.print(" and " + i);
        }
    }
    // end the line of output
    System.out.println();
}
```

```java
// Prints the factors of an integer separated by " and ".
// Precondition: n >= 1
public static void printFactors(int n) {
    // print all factors other than n, if any, followed by " and "
    for (int i = 1; i <= n - 1; i++) {
        if (n % i == 0) {
            System.out.print(i + " and ");
        }
    }
    // print last factor, n itself (fencepost)
    System.out.println(n);
}
```

10.

```java
public static void hopscotch(int hops) {
    System.out.println("   1");
    for (int i = 1; i <= hops; i++) {
        System.out.println((3 * i - 1) + "     " + 3 * i);
        System.out.println("   " + (3 * i + 1));
    }
}
```

```java
public static void hopscotch(int hops) {
    for (int i = 0; i < hops; i++) {
        System.out.println("   " + (3 * i + 1));
        System.out.println((3 * i + 2) + "     " + (3 * i + 3));
    }
    System.out.println("   " + (3 * hops + 1));
}
```

```java
public static void hopscotch(int hops) {
    for (int i = 1; i <= hops * 3 + 1; i++) {
        if (i % 3 == 1) {
            System.out.println("   " + i);
        } else if (i % 3 == 2) {
            System.out.print(i);
        } else {
            System.out.println("     " + i);
        }
    }
}
```

```java
public static void hopscotch(int hops) {
    System.out.println("   1");
    int num = 2;
    for (int i = 1; i <= hops; i++) {
        System.out.println(num + "     " + (num + 1));
        System.out.println("   " + (num + 2));
        num = num + 3;
    }
}
```

```
public static void hopscotch(int hops) {
    System.out.println("   1");
    int num = 2;
    while (num <= 3 * hops + 1) {
        System.out.print(num);
        num++;
        System.out.println("     " + num);
        num++;
        System.out.println("   " + num);
        num++;
    }
}
```

```
public static void hopscotch(int hops) {
    int count = 1;
    for (int i = 1; i <= 2 * hops + 1; i++) {
        if (i % 2 == 1) {
            System.out.println("   " + count);
            count++;
        } else {
            System.out.println(count + "     " + (count + 1));
            count += 2;
        }
    }
}
```

```
public static void hopscotch(int hops) {
    int num = 1;
    System.out.println("   " + num++);
    for (int i = 1; i <= hops; i++) {
        System.out.println(num++ + "     " + num++ + "\n   " + num++);
    }
}
```

11.

```
public static void threeHeads() {
    Random rand = new Random();
    int heads = 0;
    while (heads < 3) {
        int flip = rand.nextInt(2);  // flip coin
        if (flip == 0) {             // heads
            heads++;
            System.out.print("H ");
        } else {                     // tails
            heads = 0;
            System.out.print("T ");
        }
    }
    System.out.println();
    System.out.println("Three heads in a row!");
}
```

```
public static void threeHeads() {
    Random r = new Random();
    int heads = 0;
    do {
        if (r.nextBoolean()) {    // tails
            heads = 0;
            System.out.print("T ");
        } else {                  // heads
            heads++;
            System.out.print("H ");
        }
    } while (heads < 3);
    System.out.println("\nThree heads in a row!");
}
```

12.

```
public static void printAverage() {
    System.out.print("Type a number: ");
    Scanner console = new Scanner(System.in);
    int input = console.nextInt();
    double sum = 0.0;
    int count = 0;
    while (input >= 0) {
        count++;
        sum = sum + input;
        System.out.print("Type a number: ");
        input = console.nextInt();
    }
    if (count > 0) {
        double average = sum / count;
        System.out.println("Average was " + average);
    }
}
```

13.

```
public static boolean consecutive(int a, int b, int c) {
    return (a + 1 == b && b + 1 == c) ||
           (a + 1 == c && c + 1 == b) ||
           (b + 1 == a && a + 1 == c) ||
           (b + 1 == c && c + 1 == a) ||
           (c + 1 == a && a + 1 == b) ||
           (c + 1 == b && b + 1 == a);
}
```

```
public static boolean consecutive(int a, int b, int c) {
    int min = Math.min(a, Math.min(b, c));
    int max = Math.max(a, Math.max(b, c));
    int mid = a + b + c - max - min;
    return min + 1 == mid && mid + 1 == max;
}
```

14.

```
public static boolean hasMidpoint(int a, int b, int c) {
    double mid = (a + b + c) / 3.0;
    return (a == mid || b == mid || c == mid);
}
```

```
public static boolean hasMidpoint(int a, int b, int c) {
    double mid = (a + b + c) / 3.0;
    if (a == mid || b == mid || c == mid) {
        return true;
    } else {
        return false;
    }
}
```

```
public static boolean hasMidpoint(int a, int b, int c) {
    return (a == (b + c) / 2.0 || b == (a + c) / 2.0 || c == (a + b) / 2.0);
}
```

```
public static boolean hasMidpoint(int a, int b, int c) {
    int max = Math.max(a, Math.max(b, c));
    int min = Math.min(a, Math.min(b, c));
    double mid = (max + min) / 2.0;
    return (a == mid || b == mid || c == mid);
}
```

```
public static boolean hasMidpoint(int a, int b, int c) {
    return (a - b == b - c || b - a == a - c || a - c == c - b);
}
```

15.

```
public static boolean dominant(int a, int b, int c) {
    return a > b + c || b > a + c || c > a + b;
}
```

```
public static boolean dominant(int a, int b, int c) {
    if (a + b < c) {
        return true;
    } else if (a + c < b) {
        return true;
    } else if (b + c < a) {
        return true;
    } else {
        return false;
    }
}
```

```
public static boolean dominant(int a, int b, int c) {
    if (c > a + b) {
        return true;
    }
    if (b > a + c) {
        return true;
    }
    if (a > b + c) {
        return true;
    }
    return false;
}
```

```
public static boolean dominant(int a, int b, int c) {
    int max = Math.max(a, Math.max(b, c));
    int min = Math.min(a, Math.min(b, c));
    int mid = a + b + c - max - min;
    if (max > min + mid) {
        return true;
    } else {
        return false;
    }
}
```

```
public static boolean dominant(int a, int b, int c) {
    int sum = a + b + c;
    int max = Math.max(a, Math.max(b, c));
    return sum - max < max;
}
```

```
public static boolean dominant(int a, int b, int c) {
    int max = Math.max(a, Math.max(b, c));
    int min = Math.min(a, Math.min(b, c));
    int sum = Math.min(Math.min(a+b, a+c), b+c);
    return a > sum || b > sum || c > sum;
}
```

16.

```
// nested ifs with || solution
public static boolean anglePairs(int a1, int a2, int a3) {
    if (a1 + a2 == 90 || a2 + a3 == 90 || a3 + a1 == 90) {
        if (a1 + a2 == 180 || a2 + a3 == 180 || a3 + a1 == 180) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

```
// nest by which pair matches 90 vs. 180
public static boolean anglePairs(int a1, int a2, int a3) {
    if (a1 + a2 == 90) {
        if (a2 + a3 == 180 || a1 + a3 == 180) {
            return true;
        } else {
            return false;
        }
    } else if (a2 + a3 == 90) {
        if (a1 + a2 == 180 || a1 + a3 == 180) {
            return true;
        } else {
            return false;
        }
    } else if (a1 + a3 == 90) {
        if (a1 + a2 == 180 || a2 + a3 == 180) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

```
// boolean zen solution
public static boolean anglePairs(int a1, int a2, int a3) {
    return (a1 + a2 == 90 || a2 + a3 == 90 || a3 + a1 == 90) &&
           (a1 + a2 == 180 || a2 + a3 == 180 || a3 + a1 == 180);
}
```

17.

```
public static boolean monthApart(int m1, int d1, int m2, int d2) {
    if (m1 < m2 - 1 || m1 > m2 + 1) {        // more than one month apart
        return true;
    } else if (m1 == m2 - 1 && d1 <= d2) {   // one month apart, days far
        return true;
    } else if (m1 == m2 + 1 && d1 >= d2) {   // one month apart, days far
        return true;
    } else {                                 // same month
        return false;
    }
}
```

```
public static boolean monthApart(int m1, int d1, int m2, int d2) {
    if (m1 == m2) {
        return false;
    } else if (m1 <= m2 - 2) {
        return true;
    } else if (m1 >= m2 + 2) {
        return true;
    } else if (m1 == m2 - 1) {
        if (d1 <= d2) {
            return true;
        } else {
            return false;
        }
    } else if (m1 == m2 + 1) {
        if (d1 >= d2) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

```
public static boolean monthApart(int m1, int d1, int m2, int d2) {
    return (m2 - m1 > 1) || (m1 - m2 > 1) ||
           (m2 - m1 == 1 && d1 <= d2) ||
         (m1 - m2 == 1 && d1 >= d2);
}
```

```
public static boolean monthApart(int m1, int d1, int m2, int d2) {
    return Math.abs((m1 * 31 + d1) - (m2 * 31 + d2)) >= 31;
}
```

18.

```
public static int digitSum(int n) {
    n = Math.abs(n);          // handle negatives
    int sum = 0;
    while (n > 0) {
        sum = sum + (n % 10); // add last digit to sum
        n = n / 10;           // remove last digit
    }
    return sum;
}
```

19.

```
public static int firstDigit(int num) {
    if (num < 0) {
        num = num * -1;
    }
    while (num >= 10) {
        num = num / 10;
    }
    return num;
}
```

20.

```
public static int digitRange(int n) {
    n = Math.abs(n);          // handle negatives
    int min = n % 10;
    int max = n % 10;
    while (n != 0) {
        int digit = n % 10;
        min = Math.min(min, digit);
        max = Math.max(max, digit);
        n = n / 10;
    }
    return max - min + 1;
}
```

21.

```
public static int swapDigitPairs(int n) {
    int result = 0;        // accumulate the result to return in here
    int power = 1;
    while (n / 10 != 0) {
        int right = n % 10;
        int left = n / 10 % 10;
        n = n / 100;
        result += left * power + right * power * 10;
        power *= 100;
    }
    result += n * power;   // leftmost odd remaining digit, if any
    return result;
}
```

22.

```
public static boolean allDigitsOdd(int n) {
    if (n == 0) {
        return false;
    }

    while (n != 0) {
        if (n % 2 == 0) {   // check whether last digit is odd
            return false;
        }
        n = n / 10;
    }
    return true;
}
```

23.

```
public static boolean hasAnOddDigit(int n) {
    while (n != 0) {
        if (n % 2 != 0) {   // check whether last digit is odd
            return true;
        }
        n = n / 10;
    }
    return false;
}
```

24.

```
public static boolean isAllVowels(String s) {
    for (int i = 0; i < s.length(); i++) {
        String letter = s.substring(i, i + 1);
        if (!isVowel(letter)) {
            return false;
        }
    }
    return true;
}

public static boolean isVowel(String s) {
    return s.equalsIgnoreCase("a") || s.equalsIgnoreCase("e") ||
    s.equalsIgnoreCase("i") || s.equalsIgnoreCase("o") ||
    s.equalsIgnoreCase("u");
}
```

```
public static boolean isAllVowels(String s) {
    s = s.toLowerCase();
    for (int i = 0; i < s.length(); i++) {
        if (s.charAt(i) != 'a' && s.charAt(i) != 'e' && s.charAt(i) != 'i' && s.charAt(i) != 'o' && s.charAt(i) != 'u') {
            return false;  // found a non-vowel
        }
    }
    return true;       // didn't find any non-vowels
}
```

```
public static boolean isAllVowels(String s) {
    s = s.toLowerCase();
    for (int i = 0; i < s.length(); i++) {
        if ("aeiouAEIOU".indexOf(s.charAt(i)) < 0) {
            return false;   // found a non-vowel
        }
    }
    return true;       // didn't find any non-vowels
}
```

Chapter 6

1.

```
public static void boyGirl(Scanner input) {
    int boys = 0;
    int girls = 0;
    int boySum = 0;
    int girlSum = 0;
    while (input.hasNext()) {
        String throwAway = input.next();  // throw away name
        if (boys == girls) {
            boys++;
            boySum += input.nextInt();
        } else {
            girls++;
            girlSum += input.nextInt();
        }
    }
    System.out.println(boys + " boys, " + girls + " girls");
    System.out.println("Difference between boys' and girls' sums: " + Math.abs(boySum - girlSum));
}
```

```
public static void boyGirl(Scanner input) {
    int count = 0;  // number of people seen
    int diff = 0;   // difference between boys' and girls' sum
    while (input.hasNext()) {
        input.next();  // throw away name
        count++;
        if (count % 2 == 1) {
            diff += input.nextInt();
        } else {
            diff -= input.nextInt();
        }
    }
    System.out.println((count + 1) / 2 + " boys, " + (count / 2) + " girls");
    System.out.println("Difference between boys' and girls' sums: " + Math.abs(diff));
}
```

2.

```
public static void evenNumbers(Scanner input) {
    int count = 0;
    int evens = 0;
    int sum = 0;
    while (input.hasNextInt()) {
        int number = input.nextInt();
        count++;
        sum += number;
        if (number % 2 == 0) {
            evens++;
        }
    }
    double percent = 100.0 * evens / count;
    System.out.println(count + " numbers, sum = " + sum);
    System.out.printf("%d evens (%.2f%%)\n", evens, percent);
}
```

3.

```
public static boolean negativeSum(Scanner input) {
    int sum = 0;
    int count = 0;
    while (input.hasNextInt()) {
        int next = input.nextInt();
        sum += next;
        count++;
        if (sum < 0) {
            System.out.println(sum + " after " + count + " steps");
            return true;
        }
    }
    System.out.println("no negative sum");
    return false;  // not found
}
```

4.

```
// count up money as a double of dollars/cents; use printf at end
public static void countCoins(Scanner input) {
    double total = 0.0;
    while (input.hasNext()) {
        int count = input.nextInt();
        String coin = input.next().toLowerCase();
        if (coin.equals("nickels")) {
            count = count * 5;
        } else if (coin.equals("dimes")) {
            count = count * 10;
        } else if (coin.equals("quarters")) {
            count = count * 25;
        }
        total = total + (double) count / 100;
    }
    System.out.printf("Total money: $%.2f\n", total);
}
```

5.

```
public static void collapseSpaces(Scanner input) {
    while (input.hasNextLine()) {
        String text = input.nextLine();
        Scanner words = new Scanner(text);
        if (words.hasNext()) {
            String word = words.next();
            System.out.print(word);
            while (words.hasNext()) {
                word = words.next();
                System.out.print(" " + word);
            }
        }
        System.out.println();
    }
}
```

6.

```
public static String readEntireFile(Scanner input) {
    String text = "";
    while (input.hasNextLine()) {
        text += input.nextLine() + "\n";
    }
    return text;
}
```

7.

```
public static void flipLines(Scanner input) {
    while (input.hasNextLine()) {
        String first = input.nextLine();
        if (input.hasNextLine()) {
            String second = input.nextLine();
            System.out.println(second);
        }
        System.out.println(first);
    }
}
```

8.

```
public static void doubleSpace(Scanner in, PrintStream out) {
    while (in.hasNextLine()) {
        out.println(in.nextLine());
        out.println();
    }
    out.close();
}
```

```
public static void doubleSpace(Scanner in, PrintStream out) {
    // read the input file and store as a long String
    String output = "";
    while (in.hasNextLine()) {
        output = output + in.nextLine() + "\n\n";
    }
    // write the doubled output to the outFile
    out.print(output);
    out.close();
}
```

9.

```
public static void wordWrap(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        while (line.length() > 60) {
            String first60 = line.substring(0, 60);
            System.out.println(first60);
            line = line.substring(60);
        }
        System.out.println(line);
    }
}
```

10.

```
public static void wordWrap2(Scanner input, PrintStream out) {
    int max = 60;
    while (input.hasNextLine()) {
        String line = input.nextLine();
        while (line.length() > max) {
            String first = line.substring(0, max);
            out.println(first);
            line = line.substring(max);
        }
        out.println(line);
    }
    out.close();
}
```

11.

```
public static void wordWrap3(Scanner input) throws FileNotFoundException {
    int max = 60;
    while (input.hasNextLine()) {
        String line = input.nextLine();
        while (line.length() > max) {
            // find the nearest token boundary
            int index = max;
            while (!Character.isWhitespace(line.charAt(index))) {
                index--;
            }
            String first = line.substring(0, index + 1);
            System.out.println(first);
            line = line.substring(index + 1);
        }
        System.out.println(line);
    }
}
```

12.

```
public static void stripHtmlTags(Scanner input) throws FileNotFoundException {
    String text = "";
    while (input.hasNextLine()) {
        text += input.nextLine() + "\n";
    }
    int indexOfTag = text.indexOf("<");
    while (indexOfTag >= 0) {
        String start = text.substring(0, indexOfTag);
        text = text.substring(indexOfTag, text.length());
        int indexOfTagEnd = text.indexOf(">");
        text = start + text.substring(indexOfTagEnd + 1, text.length());
        indexOfTag = text.indexOf("<");
    }
    System.out.print(text);
}
```

13.

```
public static void stripComments(Scanner input) throws FileNotFoundException {
    String text = "";
    while (input.hasNextLine()) {
        text += input.nextLine() + "\n";
    }
    int index1 = text.indexOf("//");
    int index2 = text.indexOf("/*");
    while (index1 >= 0 || index2 >= 0) {
        if (index2 < 0 || (index1 >= 0 && index1 < index2)) {
            String start = text.substring(0, index1);
            text = text.substring(index1, text.length());
            text = start + text.substring(text.indexOf("\n"), text.length());
        } else {
            String start = text.substring(0, index2);
            text = text.substring(index2, text.length());
            text = start + text.substring(text.indexOf("*/") + 2, text.length());
        }
        index1 = text.indexOf("//");
        index2 = text.indexOf("/*");
    }
    System.out.print(text);
}
```

14.

```
public static void printDuplicates(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        String token = lineScan.next();
        int count = 1;
        while (lineScan.hasNext()) {
            String token2 = lineScan.next();
            if (token2.equals(token)) {
                count++;
            } else {
                if (count > 1) {
                    System.out.print(token + "*" + count + " ");
                }
                token = token2;
                count = 1;
            }
        }
        if (count > 1) {
            System.out.print(token + "*" + count);
        }
        System.out.println();
    }
}
```

```
public static void printDuplicates(Scanner input) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScan = new Scanner(line);
        String token = lineScan.next();
        int count = 1;
        while (lineScan.hasNext()) {
            String token2 = lineScan.next();
            if (token2.equals(token)) {
                count++;
            }
            if (count > 1 && (!lineScan.hasNext() || !token2.equals(token))) {
                System.out.print(token + "*" + count + " ");
                count = 1;
            }
            token = token2;
        }
        System.out.println();
    }
}
```

15.

```
public static void coinFlip(Scanner input) {
    while (input.hasNextLine()) {
        Scanner lineScan = new Scanner(input.nextLine().toUpperCase());
        int heads = 0;
        int total = 0;
        while (lineScan.hasNext()) {
            total++;
            if (lineScan.next().equals("H")) {
                heads++;
            }
        }

        double percent = 100.0 * heads / total;
        System.out.printf("%d heads (%.1f%%)\n", heads, percent);
        if (percent > 50) {
            System.out.println("You win!");
        }
        System.out.println();
    }
}
```

16.

```
public static int mostCommonNames(Scanner input) {
    int totalCount = 0;
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner words = new Scanner(line);

        String mostCommon = words.next();
        int mostCount = 1;
        String current = mostCommon;
        int currentCount = 1;
        totalCount++;

        while (words.hasNext()) {
            String next = words.next();
            if (next.equals(current)) {
                currentCount++;
                if (currentCount > mostCount) {
                    mostCount = currentCount;
                    mostCommon = current;
                }
            } else {
                current = next;
                currentCount = 1;
                totalCount++;
            }
        }
        System.out.println("Most common: " + mostCommon);
    }
    return totalCount;
}
```

17.

```
public static void inputStats(Scanner input) {
    int lines = 0;
    String longestLine = "";

    while (input.hasNextLine()) {
        String line = input.nextLine();
        lines++;
        if (line.length() > longestLine.length()) {
            longestLine = line;
        }

        Scanner lineScan = new Scanner(line);
        int tokens = 0;
        int longest = 0;
        while (lineScan.hasNext()) {
            String token = lineScan.next();
            tokens++;
            longest = Math.max(longest, token.length());
        }

        System.out.println("Line " + lines + " has " + tokens +
        " tokens (longest = " + longest + ")");
    }
    System.out.println("Longest line: " + longestLine);
}
```

18.

```
public static void plusScores(Scanner input) {
    while (input.hasNextLine()) {
        String name = input.nextLine();
        String data = input.nextLine();
        int plus = 0;
        int count = 0;
        for (int i = 0; i < data.length(); i++) {
            count++;
            if (data.charAt(i) == '+') {
                plus++;
            }
        }
        double percent = 100.0 * plus / count;
        System.out.println(name + ": " + percent + "% plus");
    }
}
```

19.

```
public static void leetSpeak(Scanner input, PrintStream output) {
    while (input.hasNextLine()) {
        String line = input.nextLine();
        Scanner lineScanner = new Scanner(line);
        while (lineScanner.hasNext()) {
            String word = lineScanner.next();
            word = word.replace("o", "0");
            word = word.replace("l", "1");
            word = word.replace("e", "3");
            word = word.replace("a", "4");
            word = word.replace("t", "7");
            if (word.endsWith("s")) {
                word = word.substring(0, word.length() - 1) + "Z";
            }
            output.print("(" + word + ") ");
        }
        output.println();
    }
}
```

Chapter 7

1.

```
public static int lastIndexOf(int[] a, int target) {
    for (int i = a.length - 1; i >= 0; i--) {
        if (a[i] == target) {
            return i;
        }
    }
    return -1;
}
```

2.

```
public static int range(int[] a) {
    int min = 0;
    int max = 0;
    for (int i = 0; i < a.length; i++) {
        if (i == 0 || a[i] < min) {
            min = a[i];
        }
        if (i == 0 || a[i] > max) {
            max = a[i];
        }
    }
    int valueRange = max - min + 1;
    return valueRange;
}
```

```
public static int range(int[] a) {
    int min = a[0];
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
        min = Math.min(min, a[i]);
        max = Math.max(max, a[i]);
    }
    return max - min + 1;
}
```

```
public static int range(int[] a) {
    int[] copy = new int[a.length];
    for (int i = 0; i < a.length; i++) {
        copy[i] = a[i];
    }
    Arrays.sort(copy);
    return copy[copy.length - 1] - copy[0] + 1;
}
```

```
public static int range(int[] a) {
    int range = 1;
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a.length; j++) {
            int difference = Math.abs(a[i] - a[j]) + 1;
            if (difference > range) {
                range = difference;
            }
        }
    }
    return range;
}
```

3.

```
public static int countInRange(int[] a, int min, int max) {
    int count = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] >= min && a[i] <= max) {
            count++;
        }
    }
    return count;
}
```

4.

```
public static boolean isSorted(double[] list) {
    for (int i = 0; i < list.length - 1; i++) {
        if (list[i] > list[i + 1]) {
            return false;
        }
    }
    return true;
}
```

5.

```
public static int mode(int[] a) {
    // tally all the occurrences of each element
    int[] tally = new int[101];
    for (int i = 0; i < a.length; i++) {
        tally[a[i]]++;
    }
    // scan the array of tallies to find the highest tally (the mode)
    int maxCount = 0;
    int modeValue = 0;
    for (int i = 0; i < tally.length; i++) {
        if (tally[i] > maxCount) {
            maxCount = tally[i];
            modeValue = i;
        }
    }
    return modeValue;
}
```

6.

```
public static double stdev(int[] a) {
    if (a.length == 1) {
        return 0.0;
    }
    int sum = 0;
    for (int i = 0; i < a.length; i++) {
        sum += a[i];
    }
    double average = (double) sum / a.length;
    double sumDiff = 0.0;
    for (int i = 0; i < a.length; i++) {
        sumDiff += Math.pow(a[i] - average, 2);
    }
    return Math.abs(Math.sqrt(sumDiff / (a.length - 1)));
}
```

7.

```
public static int kthLargest(int k, int[] a) {
    int[] a2 = new int[a.length];
    for (int i = 0; i < a.length; i++) {
        a2[i] = a[i];
    }
    Arrays.sort(a2);
    return a2[a2.length - 1 - k];
}
```

```
public static int kthLargest(int k, int[] a) {
    int[] a2 = Arrays.copyOf(a, a.length);
    Arrays.sort(a2);
    return a2[a2.length - 1 - k];
}
```

8.

```
public static int median(int[] a) {
    // count the number of occurrences of each number into a "tally" array
    int[] tally = new int[100];
    for (int i = 0; i < a.length; i++) {
        tally[a[i]]++;
    }
    // examine the tallies and stop when we have seen half the numbers
    int i;
    for (i = 0; tally[i] <= a.length / 2; i++) {
        tally[i + 1] += tally[i];
    }
    return i;
}
```

9.

```
public static int minGap(int[] list) {
    if (list.length < 2) {
        return 0;
    } else {
        int min = list[1] - list[0];
        for (int i = 2; i < list.length; i++) {
            int gap = list[i] - list[i - 1];
            if (gap < min) {
                min = gap;
            }
        }
        return min;
    }
}
```

10.

```java
public static double percentEven(int[] list) {
    if (list.length == 0) {
        return 0.0;
    }
    int numEven = 0;
    for (int element: list) {
        if (element % 2 == 0) {
            numEven++;
        }
    }
    return numEven * 100.0 / list.length;
}
```

```java
public static double percentEven(int[] list) {
    if (list.length == 0) {
        return 0.0;
    }
    int numEven = 0;
    for (int i = 0; i < list.length; i++) {
        if (list[i] % 2 == 0) {
            numEven++;
        }
    }
    return numEven * 100.0 / list.length;
}
```

11.

```java
public static boolean isUnique(int[] list) {
    for (int i = 0; i < list.length; i++) {
        for (int j = i + 1; j < list.length; j++) {
            if (list[i] == list[j]) {
                return false;
            }
        }
    }
    return true;
}
```

12.

```
public static int priceIsRight(int[] bids, int price) {
    int bestPrice = -1;
    for (int i = 0; i < bids.length; i++) {
        if (bids[i] <= price && bids[i] > bestPrice) {
            bestPrice = bids[i];
        }
    }
    return bestPrice;
}
```

```
public static int priceIsRight(int[] prices, int price) {
    int bestPrice = -1;
    for (int i = 0; i < prices.length; i++) {
        if (prices[i] <= price && prices[i] > bestPrice) {
            bestPrice = prices[i];
        }
    }
    if (bestPrice <= price) {
        return bestPrice;
    } else {
        return -1;
    }
}
```

```
public static int priceIsRight(int[] bids, int price) {
    int[] difference = new int[bids.length];
    int bestAnswer = Integer.MIN_VALUE;
    for (int i = 0; i < difference.length; i++) {
        difference[i] = bids[i] - price;
    }
    for (int i = 0; i < difference.length; i++) {
        if (difference[i] <= 0) {
            bestAnswer = (int) Math.max(difference[i], bestAnswer);
        }
    }
    if (bestAnswer == Integer.MIN_VALUE) {
        return -1;
    } else {
        return bestAnswer + price;
    }
}
```

13.

```
public static int longestSortedSequence(int[] list) {
    if (list.length == 0) {
        return 0;
    }
    int max = 1;
    int count = 1;
    for (int i = 1; i < list.length; i++) {
        if (list[i] >= list[i - 1]) {
            count++;
        } else {
            count = 1;
        }
        if (count > max) {
            max = count;
        }
    }
    return max;
}
```

14.

```
public static boolean contains(int[] a1, int[] a2) {
    for (int i = 0; i <= a1.length - a2.length; i++) {
        boolean found = true;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i + j] != a2[j]) {
                found = false;
            }
        }
        if (found) {
            return true;
        }
    }
    return false;
}
```

```
// varation of first solution that uses count instead of boolean
public static boolean contains(int[] a1, int[] a2) {
    for (int i = 0; i <= a1.length - a2.length; i++) {
        int count = 0;
        for (int j = 0; j < a2.length; j++) {
            if (a1[i + j] == a2[j])
                count++;
        }
        if (count == a2.length)
            return true;
    }
    return false;
}
```

```
public static boolean contains(int[] a1, int[] a2) {
    int i1 = 0;
    int i2 = 0;
    while (i1 < a1.length && i2 < a2.length) {
        if (a1[i1] != a2[i2]) {  // does not match, starts over
            i2 = 0;
        }
        if (a1[i1] == a2[i2]) {
            i2++;
        }
        i1++;
    }
    return i2 >= a2.length;
}
```

```
public static boolean contains(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        int j = 0;
        while (j < a2.length && i + j < a1.length && a1[i + j] == a2[j])
            j++;
        if (j == a2.length)
            return true;
    }
    return false;
}
```

15.

```
public static int[] collapse(int[] list) {
    int[] result = new int[list.length / 2 + list.length % 2];
    for (int i = 0; i < result.length - list.length % 2; i++) {
        result[i] = list[2 * i] + list[2 * i + 1];
    }
    if (list.length % 2 == 1) {
        result[result.length - 1] = list[list.length - 1];
    }
    return result;
}
```

16.

```
public static int[] append(int[] list1, int[] list2) {
    int[] result = new int[list1.length + list2.length];
    for (int i = 0; i < list1.length; i++) {
        result[i] = list1[i];
    }
    for (int i = 0; i < list2.length; i++) {
        result[i + list1.length] = list2[i];
    }
    return result;
}
```

17.

```
public static int[] vowelCount(String text) {
    int[] counts = new int[5];
    for (int i = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        if (c == 'a') {
            counts[0]++;
        } else if (c == 'e') {
            counts[1]++;
        } else if (c == 'i') {
            counts[2]++;
        } else if (c == 'o') {
            counts[3]++;
        } else if (c == 'u') {
            counts[4]++;
        }
    }
    return counts;
}
```

```
public static int[] vowelCount(String text) {
    int[] counts = new int[5];
    for (int i = 0; i < text.length(); i++) {
        int index = "aeiou".indexOf(text.charAt(i));
        if (index >= 0) {
            counts[index]++;
        }
    }
    return counts;
}
```

18.

```
public static void wordLengths(String filename) throws FileNotFoundException {
    // tally the lengths of every word in the file
    Scanner input = new Scanner(new File(filename));
    int[] tally = new int[81];
    int maxLength = 0;
    while (input.hasNext()) {
        String token = input.next();
        tally[token.length()]++;
        maxLength = Math.max(maxLength, token.length());
    }
    // report the results
    for (int i = 1; i <= maxLength; i++) {
        if (tally[i] > 0) {
            System.out.print(i + ": " + tally[i] + "\t");
            for (int j = 0; j < tally[i]; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

19.

```
public static int[][] matrixAdd(int[][] a, int[][] b) {
    int rows = a.length;
    int cols = 0;
    if (rows > 0) {
        cols = a[0].length;
    }
    int[][] c = new int[rows][cols];
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```

20.

```
public static boolean isMagicSquare(int[][] a) {
    if (a.length == 0 || a.length == 1 && a[0].length == 1) {
        return true;   // trivial cases; empty and one-element array
    }

    // get sum of first row, use as basis for others
    int sum = 0;
    for (int c = 0; c < a[0].length; c++) {
        sum += a[0][c];
    }

    // compute all row sums
    for (int r = 0; r < a.length; r++) {
        if (a[r].length != a.length) {
            return false;   // not square shape
        }
        int rowSum = 0;
        for (int c = 0; c < a[r].length; c++) {
            rowSum += a[r][c];
        }
        if (rowSum != sum) {
            return false;
        }
    }

    // compute all column sums
    for (int c = 0; c < a.length; c++) {
        int colSum = 0;
        for (int r = 0; r < a.length; r++) {
            colSum += a[r][c];
        }
        if (colSum != sum) {
            return false;
        }
    }

    // compute diagonal sums
    int diagSum1 = 0;
    int diagSum2 = 0;
    for (int rc = 0; rc < a.length; rc++) {
        diagSum1 += a[rc][rc];
        diagSum2 += a[rc][a.length - 1 - rc];
    }
    if (diagSum1 != sum || diagSum2 != sum) {
        return false;
    }

    return true;
}
```

```
public static boolean isMagicSquare(int[][] a) {
    if (a.length == 0) {
        return true;
    }
    int sum = sumHelper(a, 0, 0, 0, 1);       // sum of first row
    for (int r = 0; r < a.length; r++) {   // row sums
        if (a[r].length != a.length || sumHelper(a, r, 0, 0, 1) != sum) {
            return false;
        }
    }
    for (int c = 0; c < a.length; c++) {   // column sums
        if (sumHelper(a, 0, c, 1, 0) != sum) {
            return false;
        }
    }
    return sumHelper(a, 0, 0, 1, 1) == sum &&
    sumHelper(a, 0, a[0].length - 1, 1, -1) == sum;   // diagonal sums
}

private static int sumHelper(int[][] a, int r, int c, int dr, int dc) {
    int sum = 0;
    while (r >= 0 && c >= 0 && r < a.length && c < a[r].length) {
        sum += a[r][c];
        r += dr;
        c += dc;
    }
    return sum;
}
```

Chapter 8

1.

```
public int quadrant() {
    if (x > 0 && y > 0) {
        return 1;
    } else if (x < 0 && y > 0) {
        return 2;
    } else if (x < 0 && y < 0) {
        return 3;
    } else if (x > 0 && y < 0) {
        return 4;
    } else {
        return 0;
    }
}
```

2.

```
public void flip() {
    int temp = x;
    x = -y;
    y = -temp;
}
```

3.

```
// Returns the "Manhattan (rectangular) distance" between
// this point and the given other point.
public int manhattanDistance(Point other) {
    int dx = x - other.x;
    int dy = y - other.y;
    return Math.abs(dx) + Math.abs(dy);
}
```

4.

```
// Returns true if the given point lines up vertically
// with this point (if they have the same x value).
public boolean isVertical(Point p) {
    return x == p.x;
}
```

5.

```
public double slope(Point other) {
    if (x == other.x) {
        throw new IllegalArgumentException();
    }
    return (other.y - y) / (1.0 * other.x - x);
}
```

6.

```
public boolean isCollinear(Point p1, Point p2) {
    // basic case: all points have same x or y value
    if ((x == p1.x && x == p2.x) || (y == p1.y && y == p2.y)) {
        return true;
    }
    // complex case: compare slopes
    double slope1 = (p1.y - y) / (p1.x - x);
    double slope2 = (p2.y - y) / (p2.x - x);
    return round(slope1, 4) == round(slope2, 4);
}
private double round(double value, int places) {
    for (int i = 0; i < places; i++) {
        value *= 10.0;
    }
    value = Math.round(value);
    for (int i = 0; i < places; i++) {
        value /= 10.0;
    }
    return value;
}
```

7.

```
// Adds the amount of time represented by the given time
// span to this time span.
public void add(TimeSpan span) {
    hours += span.hours;
    minutes += span.minutes;
    hours += minutes / 60;
    minutes = minutes % 60;
}
```

```
// Adds the amount of time represented by the given time
// span to this time span.
public void add(TimeSpan time) {
    add(time.hours, time.minutes);
}
```

8.

```
public void subtract(TimeSpan span) {
    hours -= span.hours;
    minutes -= span.minutes;
    if (minutes < 0) {
        minutes = minutes + 60;
        hours--;
    }
}
```

9.

```
// Adds the given interval to this time span.
// pre: hours >= 0 and minutes >= 0
public void scale(int factor) {
    hours *= factor;
    minutes *= factor;
    // convert any overflow of 60 minutes into one hour
    hours += minutes / 60;
    minutes = minutes % 60;
}
```

10.

```
// Removes all purchases of this Stock.
public void clear() {
    totalShares = 0;
    totalCost = 0.00;
}
```

11.

```
public boolean transactionFee(double amount) {
    for (int i = 1; i <= transactions; i++) {
        balance -= amount * i;
    }
    if (balance > 0.0) {
        return true;
    } else {
        balance = 0.0;
        return false;
    }
}
```

12.

```
public String toString() {
    String result = name + ", ";

    // handle negative numbers
    if (balance < 0) {
        result += "-";
    }

    result += "$" + Math.abs(balance);

    // handle numbers with cents a multiple of ten (e.g. $12.30)
    if (result.charAt(result.length() - 2) == '.') {
        result += "0";
    }

    return result;
}
```

13.

```
public void transfer(BankAccount other, double amount) {
    if (amount > 0.00) {
        if (amount + 5.00 <= balance) {
            withdraw(amount + 5.00);
            other.deposit(amount);
        } else {
            if (amount > 5.00) {
                // partial transfer
                other.deposit(balance - 5.00);
                withdraw(balance);
            }
        }
    }
}
```

14.

```
// Represents a line segment between two Points.
public class Line {
    private Point p1;
    private Point p2;
    // Constructs a new Line that contains the given two Points.
    public Line(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
    // Returns this Line's first endpoint.
    public Point getP1() {
        return p1;
    }
    // Returns this Line's second endpoint.
    public Point getP2() {
        return p2;
    }
    // Returns a String representation of this Line, such as "[(-2, 3), (4, 7)]".
    public String toString() {
        return "[" + p1 + ", " + p2 + "]";
    }
}
```

15.

```
// Returns the slope of this Line.
public double getSlope() {
    if (p1.getX() == p2.getX()) {
        throw new IllegalArgumentException("undefined slope");
    }
    return (double) (p2.getY() - p1.getY()) /
                    (p2.getX() - p1.getX());
}
```

16.

```
// Constructs a new Line that contains the given two points.
public Line(int x1, int y1, int x2, int y2) {
    p1 = new Point(x1, y1);
    p2 = new Point(x2, y2);
}
```

```
// Constructs a new Line that contains the given two points.
public Line(int x1, int y1, int x2, int y2) {
    this(new Point(x1, y1), new Point(x2, y2));
}
```

17.

```
// Returns true if the given point is collinear with this Line.
public boolean isCollinear(Point p) {
    // basic case: all points have same x or y value
    if ((p.getX() == p1.getX() && p.getX() == p2.getX()) ||
            (p.getY() == p1.getY() && p.getY() == p2.getY())) {
        return true;
    }
    // complex case: compare slopes
    double slope1 = (p1.getY() - p.getY()) / (p1.getX() - p.getX());
    double slope2 = (p2.getY() - p.getY()) / (p2.getX() - p.getX());
    return round(slope1, 4) == round(slope2, 4);
}
// Rounds the given value to 4 digits after the decimal.
public static double round(double value, int places) {
    double pow10 = Math.pow(10, places);
    return Math.round(value * pow10) / pow10;
}
```

18.

```
// Represents a 2-dimensional rectangular region.
public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;
    // Constructs a new Rectangle whose top-left corner is specified by the
    // given coordinates and with the given width and height.
    public Rectangle(int x, int y, int width, int height) {
        if (width < 0 || height < 0) {
            throw new IllegalArgumentException();
        }
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    // Returns this Rectangle's height.
    public int getHeight() {
        return height;
    }
    // Returns this Rectangle's width.
    public int getWidth() {
        return width;
    }
    // Returns this Rectangle's x coordinate.
    public int getX() {
        return x;
    }
    // Returns this Rectangle's y coordinate.
    public int getY() {
        return y;
    }
    // Returns a String representation of this Rectangle, such as
    // "Rectangle[x=1,y=2,width=3,height=4]".
    public String toString() {
        return "Rectangle[x=" + x + ",y=" + y +
               ",width=" + width + ",height=" + height + "]";
    }
}
```

19.

```
// Constructs a new rectangle whose top-left corner is specified by the
// given point and with the given width and height.
public Rectangle(Point p, int width, int height) {
    this.x = p.x;
    this.y = p.y;
    this.width = width;
    this.height = height;
}
```

```
// Constructs a new rectangle whose top-left corner is specified by the
// given point and with the given width and height.
public Rectangle(Point p, int width, int height) {
    this(p.getX(), p.getY(), width, height);
}
```

20.

```
// Returns whether the given coordinates lie inside this Rectangle.
public boolean contains(int x, int y) {
    return this.x <= x && x < this.x + width &&
           this.y <= y && y < this.y + height;
}
// Returns whether the given point lies inside this Rectangle.
public boolean contains(Point p) {
    // return contains(p.getX(), p.getY());
    return contains(p.x, p.y);
}
```

21.

```
// Returns a new Rectangle that represents the tightest bounding box
// that contains both this rectangle and the other rectangle.
public Rectangle union(Rectangle rect) {
    int left = Math.min(x, rect.x);
    int top = Math.min(y, rect.y);
    int right = Math.max(x + width, rect.x + rect.width);
    int bottom = Math.max(y + height, rect.y + rect.height);
    return new Rectangle(left, top, right - left, bottom - top);
}
```

22.

```
// Returns a new rectangle that represents the largest rectangular region
// completely contained within both this rectangle and the given other
// rectangle.  If the rectangles do not intersect at all, returns a rectangle
// whose width and height are both 0.
public Rectangle intersection(Rectangle rect) {
    int left = Math.max(x, rect.x);
    int top = Math.max(y, rect.y);
    int right = Math.min(x + width, rect.x + rect.width);
    int bottom = Math.min(y + height, rect.y + rect.height);
    int width = Math.max(0, right - left);
    int height = Math.max(0, bottom - top);
    return new Rectangle(left, top, width, height);
}
```

Chapter 9

1.

```
// A class to represent marketers.
public class Marketer extends Employee {
    public void advertise() {
        System.out.println("Act now, while supplies last!");
    }
    public double getSalary() {
        return super.getSalary() + 10000;
    }
}
```

2.

```
// A class to represent marketers.
public class Janitor extends Employee {
    public void clean() {
        System.out.println("Workin' for the man.");
    }
    public int getHours() {
        return super.getHours() * 2;
    }
    public double getSalary() {
        return super.getSalary() - 10000.00;
    }
    public int getVacationDays() {
        return super.getVacationDays() / 2;
    }
}
```

3.

```
// A class to represent Harvard lawyers.
public class HarvardLawyer extends Lawyer {
    public double getSalary() {
        return super.getSalary() * 1.2;
    }
    public int getVacationDays() {
        return super.getVacationDays() + 3;
    }
    public String getVacationForm() {
        String lawyerForm = super.getVacationForm();
        return lawyerForm + lawyerForm + lawyerForm + lawyerForm;
    }
}
```

4.

```
public class MonsterTruck extends Truck {
    public void m1() {
        System.out.println("monster 1");
    }

    public void m2() {
        super.m1();
        super.m2();
    }

    public String toString() {
        return "monster " + super.toString();
    }
}
```

5.

```java
// Superclass for all types of tickets.
public abstract class Ticket {
    private int number;
    public Ticket(int number) {
        this.number = number;
    }
    public abstract double getPrice();
    public String toString() {
        return "Number: " + this.number +
               ", Price: " + this.getPrice();
    }
}
```

6.

```java
public class WalkupTicket extends Ticket {
    public WalkupTicket(int number) {
        super(number);
    }
    public double getPrice() {
        return 50.00;
    }
}
```

7.

```java
public class AdvanceTicket extends Ticket {
    private int daysInAdvance;
    public AdvanceTicket(int number, int daysInAdvance) {
        super(number);
        this.daysInAdvance = daysInAdvance;
    }
    public double getPrice() {
        if (daysInAdvance >= 10) {
            return 30.00;
        } else {
            return 40.00;
        }
    }
}
```

8.

```java
// Represents a student ticket purchased ahead of time.
public class StudentAdvanceTicket extends AdvanceTicket {
    public StudentAdvanceTicket(int number, int daysInAdvance) {
        super(number, daysInAdvance);
    }
    public double getPrice() {
        return super.getPrice() / 2;
    }
    public String toString() {
        return super.toString() + " (ID required)";
    }
}
```

9.

```
public class MinMaxAccount extends BankingAccount {
    private int minBalance;
    private int maxBalance;
    public MinMaxAccount(Startup s) {
        super(s);
        minBalance = getBalance();
        maxBalance = getBalance();
    }
    public void debit(Debit d) {
        super.debit(d);
        updateMinMax();
    }
    public void credit(Credit c) {
        super.credit(c);
        updateMinMax();
    }
    private void updateMinMax() {
        int balance = getBalance();
        if (balance < minBalance) {
            minBalance = balance;
        } else if(balance > maxBalance) {
            maxBalance = balance;
        }
    }
    public int getMin() {
        return minBalance;
    }
    public int getMax() {
        return maxBalance;
    }
}
```

10.

```
public class DiscountBill extends GroceryBill {
    private boolean preferred;
    private int count;
    private double discount;
    public DiscountBill(Employee clerk, boolean preferred) {
        super(clerk);
        this.preferred = preferred;
        count = 0;
        discount = 0.0;
    }
    public void add(Item i) {
        super.add(i);
        if (preferred) {
            double amount = i.getDiscount();
            if (amount > 0.0) {
                count++;
                discount += amount;
            }
        }
    }
    public double getTotal() {
        return super.getTotal() - discount;
    }
    public int getDiscountCount() {
        return count;
    }
    public double getDiscountAmount() {
        return discount;
    }
    public double getDiscountPercent() {
        return discount / super.getTotal() * 100;
    }
}
```

11.

```
public class FilteredAccount extends Account {
    private int zeros;
    private int transactions;
    public FilteredAccount(Client c) {
        super(c);
        zeros = 0;
        transactions = 0;
    }
    public boolean process(Transaction t) {
        transactions++;
        if (t.value() == 0) {
            zeros++;
            return true;
        } else {
            return super.process(t);
        }
    }
    public double percentFiltered() {
        if (transactions == 0) {
            return 0.0;
        } else {
            return zeros * 100.0 / transactions;
        }
    }
}
```

12.

```
// first implementation (hours and minutes fields)
public boolean equals(Object o) {
    if (o instanceof TimeSpan) {
        TimeSpan other = (TimeSpan) o;
        return hours == other.hours && minutes == other.minutes;
    } else {
        return false;
    }
}
// second implementation (totalMinutes field)
public boolean equals(Object o) {
    if (o instanceof TimeSpan) {
        TimeSpan other = (TimeSpan) o;
        return totalMinutes == other.totalMinutes;
    } else {
        return false;
    }
}
```

13.

```
public boolean equals(Object o) {
    if (o instanceof Cash) {
        Cash other = (Cash) o;
        return amount == other.amount;
    } else {
        return false;
    }
}
```

14.

```
// Rectangle
public boolean equals(Object o) {
    if (o instanceof Rectangle) {
        Rectangle other = (Rectangle) o;
        return width == other.width && height == other.height;
    } else {
        return false;
    }
}
// Circle
public boolean equals(Object o) {
    if (o instanceof Triangle) {
        Triangle other = (Triangle) o;
        return a == other.a && b == other.b && c == other.c;
    } else {
        return false;
    }
}
// Triangle
public boolean equals(Object o) {
    if (o instanceof Rectangle) {
        Rectangle other = (Rectangle) o;
        return width == other.width && height == other.height;
    } else {
        return false;
    }
}
```

15.

```
public class Octagon implements Shape {
    private double sideLength;

    public Octagon(double sideLength) {
        this.sideLength = sideLength;
    }

    // formula taken from http://mathworld.wolfram.com/Octagon.html
    public double getArea() {
        return 2 * (1 + Math.sqrt(2)) * sideLength * sideLength;
    }

    public double getPerimeter() {
        return 8 * sideLength;
    }
}
```

16.

```
public class Hexagon implements Shape {
    private double sideLength;

    public Hexagon(double sideLength) {
        this.sideLength = sideLength;
    }

    // formula taken from http://mathworld.wolfram.com/Hexagon.html
    public double getArea() {
        return 1.5 * Math.sqrt(3) * sideLength * sideLength;
    }

    public double getPerimeter() {
        return 6 * sideLength;
    }
}
```

17.

```
// An interface to represent items storing a value that can be incremented.
public interface Incrementable {
    public void increment();
    public int getValue();
}
```

```
// A class that stores a value that increases by 1
// each time the increment method is called.
public class SequentialIncrementer implements Incrementable {
    private int value;
    // Constructs a new incrementer with value 0.
    public SequentialIncrementer() {
        value = 0;
    }
    // Increases the value by 1.
    public void increment() {
        value++;
    }
    // Returns this incrementer's current value.
    public int getValue() {
        return value;
    }
}
```

```
// A class that stores a random value that changes
// each time the increment method is called.
import java.util.*;  // for Random
public class RandomIncrementer implements Incrementable {
    private int value;
    private Random rand;
    // Constructs a new incrementer with value 0.
    public RandomIncrementer() {
        rand = new Random();
        increment();
    }
    // Increases the value by 1.
    public void increment() {
        value = rand.nextInt();
    }
    // Returns this incrementer's current value.
    public int getValue() {
        return value;
    }
}
```

Chapter 10

1.

```
public static double averageVowels(ArrayList<String> list) {
    int count = 0;
    for (int i = 0; i < list.size(); i++) {
        String s = list.get(i).toLowerCase();
        for (int j = 0; j < s.length(); j++) {
            char c = s.charAt(j);
            if (c == 'a' || c == 'e' || c == 'i' ||
                    c == 'o' || c == 'u') {
                count++;
            }
        }
    }
    return (double) count / list.size();
}
```

2.

```
public void swapPairs(ArrayList<String> list) {
    for (int i = 0; i < list.size() - 1; i += 2) {
        String first = list.remove(i);
        list.add(i + 1, first);
    }
}
```

```
public void swapPairs(ArrayList<String> list) {
    int i = 0;
    while (i < list.size() - 1) {
        String first = list.get(i);
        list.set(i, list.get(i + 1));
        list.set(i + 1, first);
        i += 2;
    }
}
```

3.

```
public void removeEvenLength(ArrayList<String> list) {
    int i = 0;
    while (i < list.size()) {
        if (list.get(i).length() % 2 == 0) {
            list.remove(i);
        } else {
            i++;
        }
    }
}
```

4.

```
public void doubleList(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i += 2) {
        list.add(i, list.get(i));
    }
}
```

5.

```
public static void scaleByK(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        int k = list.get(i);
        list.remove(i);
        for (int j = 0; j < k; j++) {
            list.add(i, k);
        }
    }
}
```

6.

```
public void minToFront(ArrayList<Integer> list) {
    int min = 0;
    for (int i = 1; i < list.size(); i++){
        if (list.get(i) < list.get(min)) {
            min = i;
        }
    }
    list.add(0, list.remove(min));
}
```

7.

```
public void removeDuplicates(ArrayList<String> list) {
    int index = 0;
    while (index < list.size() - 1) {
        String s1 = list.get(index);
        String s2 = list.get(index + 1);
        if (s1.equals(s2)) {
            list.remove(index + 1);
        } else {
            index++;
        }
    }
}
```

8.

```
public static void removeZeroes(ArrayList<Integer> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        if (list.get(i) == 0) {
            list.remove(i);
        }
    }
}
```

9.

```
public static int rangeBetweenZeroes(ArrayList<Integer> list) {
    int minIndex = list.size();
    int maxIndex = -1;
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) == 0) {
            minIndex = Math.min(minIndex, i);
            maxIndex = Math.max(maxIndex, i);
        }
    }

    int range = maxIndex - minIndex + 1;
    return Math.max(0, range);
}
```

10.

```
public static void removeInRange(ArrayList<String> list, String min, String max) {
    for (int i = list.size() - 1; i >= 0; i--) {
        String s = list.get(i);
        if (s.compareTo(min) >= 0 && s.compareTo(max) <= 0) {
            list.remove(i);
        }
    }
}
```

11.

```
public void stutter(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i += 2) {
        list.add(i, list.get(i));
    }
}
```

12.

```
public void markLength4(ArrayList<String> list) {
    int index = 0;
    while (index < list.size()) {
        if (list.get(index).length() == 4) {
            list.add(index, "****");
            index += 2;
        } else {
            index++;
        }
    }
}
```

13.

```
public void reverse3(ArrayList<Integer> list) {
    for (int i = 0; i < list.size() - 2; i++) {
        int temp = list.get(i);
        list.set(i, list.get(i + 2));
        list.set(i + 2, temp);
    }
}
```

14.

```
public void removeShorterStrings(ArrayList<String> list) {
    for (int i = 0; i < list.size() - 1; i++) {
        String first = list.get(i);
        String second = list.get(i + 1);
        if (first.length() <= second.length()) {
            list.remove(i);
        } else {
            list.remove(i + 1);
        }
    }
}
```

15.

```
public static void filterRange(ArrayList<Integer> list, int min, int max) {
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i) >= min && list.get(i) <= max) {
            list.remove(i);
            i--;
        }
    }
}
```

16.

```
public void clump(ArrayList<String> list) {
    for (int i = 0; i < list.size() - 1; i++) {
        String combined = "(" + list.get(i) + " " + list.get(i + 1) + ")";
        list.remove(i);
        list.remove(i);
        list.add(i, combined);
    }
}
```

17.

```
public static void interleave(ArrayList<Integer> a1, ArrayList<Integer> a2) {
    for (int i = 0; i < a2.size(); i++) {
        int index = Math.min(a1.size(), 2 * i + 1);
        a1.add(index, a2.get(i));
    }
}
```

18.

```
// A Point object represents a pair of (x, y) coordinates.
// This version implements the Comparable interface.
public class Point implements Comparable<Point> {
    private int x;
    private int y;

    // Compares this point to the given point in y-major order.
    public int compareTo(Point p) {
        if (y != p.y) {
            return y - p.y;
        } else {
            return x - p.x;
        }
    }
    // rest of class
}
```

19.

```
// first implementation (hours and minutes fields)
public class TimeSpan implements Comparable<TimeSpan> {
    private int hours;
    private int minutes;

    public int compareTo(TimeSpan ts) {
        if (hours != ts.hours) {
            return hours - ts.hours;
        } else {
            return minutes - ts.minutes;
        }
    }

    // rest of class
}
// second implementation (totalMinutes field)
public class TimeSpan implements Comparable<TimeSpan> {
    private int totalMinutes;

    public int compareTo(TimeSpan ts) {
        return totalMinutes - other.totalMinutes;
    }

    // rest of class
}
```

20.

```
public class CalendarDate implements Comparable<CalendarDate> {
    private int year;
    private int month;
    private int day;
    // Compares this calendar date to another date.
    // Dates are compared by month and then by day.
    public int compareTo(CalendarDate other) {
        if (year != other.year) {
            return year - other.year;
        } else if (month != other.month) {
            return month - other.month;
        } else {
            return day - other.day;
        }
    }

    // rest of class
}
```

Chapter 11

1.

```
// Returns a list of all prime numbers up to the given maximum
// using the Sieve of Eratosthenes algorithm.
public static LinkedList<Integer> sieve(int max) {
    LinkedList<Integer> primes = new LinkedList<Integer>();
    // add all numbers from 2 to max to a list
    LinkedList<Integer> numbers = new LinkedList<Integer>();
    // modified code
    numbers.add(2);
    for (int i = 3; i <= max; i += 2) {
        numbers.add(i);
    }

    double sqrt = Math.sqrt(max);
    while (!numbers.isEmpty()) {
        // remove a prime number from the front of the list
        int front = numbers.remove(0);
        primes.add(front);

        // modified code
        if (front >= sqrt) {
            while (!numbers.isEmpty()) {
                primes.add(numbers.remove(0));
            }
        }
        // remove all multiples of this prime number
        Iterator<Integer> itr = numbers.iterator();
        while (itr.hasNext()) {
            int current = itr.next();
            if (current % front == 0) {
                itr.remove();
            }
        }
    }

    return primes;
}
```

2.

```
public static List<Integer> alternate(List<Integer> list1, List<Integer> list2) {
    List<Integer> result = new ArrayList<Integer>();
    Iterator<Integer> i1 = list1.iterator();
    Iterator<Integer> i2 = list2.iterator();
    while (i1.hasNext() || i2.hasNext()) {
        if (i1.hasNext()) {
            result.add(i1.next());
        }
        if (i2.hasNext()) {
            result.add(i2.next());
        }
    }
    return result;
}
```

3.

```
public static void removeInRange(List<Integer> list, int value, int min, int max) {
    Iterator<Integer> itr = list.iterator();
    for (int i = 0; i < min; i++) {
        itr.next();
    }
    for (int i = min; i < max; i++) {
        if (itr.next() == value) {
            itr.remove();
        }
    }
}
```

4.

```
public static void partition(List<Integer> list, int value) {
    // partition original list into a temporary second list
    List<Integer> temp = new LinkedList<Integer>();
    Iterator<Integer> itr = list.iterator();
    while (itr.hasNext()) {
        int element = itr.next();
        if (element < value) {
            temp.add(0, element);
        } else {
            temp.add(element);
        }
    }

    // copy temp back to original list
    list.clear();
    for (Integer i : temp) {
        list.add(i);
    }
}
```

5.

```
public static void sortAndRemoveDuplicates(List<Integer> list) {
    Set<Integer> set = new TreeSet<Integer>(list);
    list.clear();
    for (Integer i : set) {
        list.add(i);
    }
}
```

6.

```
public static int countUnique(List<Integer> list) {
    Set<Integer> set = new HashSet<Integer>();
    for (int value : list) {
        set.add(value);
    }
    return set.size();
}
```

7.

```
public static int countCommon(List<Integer> list1, List<Integer> list2) {
    Set<Integer> set1 = new HashSet<Integer>(list1);
    Set<Integer> set2 = new HashSet<Integer>(list2);
    int common = 0;
    for (int value : set1) {
        if (set2.contains(value)) {
            common++;
        }
    }
    return common;
}
```

8.

```
public static int maxLength(Set<String> set) {
    int max = 0;
    for (String s : set) {
        max = Math.max(max, s.length());
    }
    return max;
}
```

9.

```
public static boolean hasOdd(Set<Integer> set) {
    for (int value : set) {
        if (value % 2 != 0) {
            return true;
        }
    }
    return false;
}
```

10.

```
public static void removeEvenLength(Set<String> set) {
    Iterator<String> itr = set.iterator();
    while (itr.hasNext()) {
        String s = itr.next();
        if (s.length() % 2 == 0) {
            itr.remove();
        }
    }
}
```

11.

```
public static Set<Integer> symmetricSetDifference(Set<Integer> set1, Set<Integer> set2) {
    Set<Integer> result = new TreeSet<Integer>();
    for (int i : set1) {
        if (!set2.contains(i)) {
            result.add(i);
        }
    }
    for (int i : set2) {
        if (!set1.contains(i)) {
            result.add(i);
        }
    }
    return result;
}
```

12.

```
public static boolean contains3(List<String> list) {
    Map<String, Integer> counts = new HashMap<String, Integer>();
    for (String value : list) {
        if (counts.containsKey(value)) {
            int count = counts.get(value);
            count++;
            counts.put(value, count);
            if (count >= 3) {
                return true;
            }
        } else {
            counts.put(value, 1);
        }
    }
    return false;
}
```

13.

```
public static boolean isUnique(Map<String, String> map) {
    Set<String> values = new HashSet();
    for (String value : map.values()) {
        if (values.contains(value)) {
            return false;   // duplicate
        } else {
            values.add(value);
        }
    }
    return true;
}
```

```
public static boolean isUnique(Map<String, String> map) {
    return new HashSet(map.values()).size() == map.values().size();
}
```

14.

```
public static Map<String, Integer> intersect(Map<String, Integer> map1, Map<String, Integer> map2) {
    Map<String, Integer> result = new TreeMap<String, Integer>();
    for (String key : map1.keySet()) {
        int value = map1.get(key);
        if (map2.containsKey(key) && value == map2.get(key)) {
            result.put(key, value);
        }
    }
    return result;
}
```

15.

```
public static int maxOccurrences(List<Integer> list) {
    Map<Integer, Integer> counts = new HashMap<Integer, Integer>();
    for (int value : list) {
        if (counts.containsKey(value)) {
            counts.put(value, counts.get(value) + 1);
        } else {
            counts.put(value, 1);
        }
    }

    int max = 0;
    for (int count : counts.values()) {
        max = Math.max(max, count);
    }
    return max;
}
```

16.

```
public static boolean is1to1(Map<String, String> map) {
    Set<String> values = new HashSet<String>();
    for (String key : map.keySet()) {
        String value = map.get(key);
        if (values.contains(value)) {
            return false;
        }
        values.add(value);
    }
    return true;
}
```

17.

```
public static boolean subMap(Map<String, String> map1, Map<String, String> map2) {
    for (String key : map1.keySet()) {
        if (!map2.containsKey(key) || !map1.get(key).equals(map2.get(key))) {
            return false;
        }
    }
    return true;
}
```

18.

```
public static Map<String, Integer> reverse(Map<Integer, String> map) {
    Map<String, Integer> result = new HashMap<String, Integer>();
    for (Integer key : map.keySet()) {
        String value = map.get(key);
        result.put(value, key);
    }
    return result;
}
```

19.

```
public static int rarest(Map<String, Integer> m) {
   if (m == null || m.isEmpty()) {
      throw new IllegalArgumentException();
   }
   Map<Integer, Integer> counts = new TreeMap<Integer, Integer>();
   for (String name : m.keySet()) {
      int age = m.get(name);
      if (counts.containsKey(age)) {
         counts.put(age, counts.get(age) + 1);
      } else {
         counts.put(age, 1);
      }
   }

   int minCount = m.size() + 1;
   int rareAge = -1;
   for (int age : counts.keySet()) {
      int count = counts.get(age);
      if (count < minCount) {
         minCount = count;
         rareAge = age;
      }
   }

   return rareAge;
}
```

20.

```
// This program reads two text files and compares the
// vocabulary used in each.
// This version uses Sets instead of Lists.
import java.util.*;
import java.io.*;
public class Vocabulary {
    public static void main(String[] args)
            throws FileNotFoundException {
        Scanner console = new Scanner(System.in);
        giveIntro();
        System.out.print("file #1 name? ");
        String filename1 = console.nextLine();
        Scanner input1 = new Scanner(new File(filename1));
        System.out.print("file #2 name? ");
        String filename2 = console.nextLine();
        Scanner input2 = new Scanner(new File(filename2));
        System.out.println();
        Set<String> set1 = getWords(input1);
        Set<String> set2 = getWords(input2);
        Set<String> overlap = new TreeSet(set1);
        overlap.retainAll(set2);
        reportResults(set1, set2, overlap);
    }
    // post: reads all words from the given Scanner, turning them to lowercase
    //       and returning a set of the vocabulary of the file
    public static Set<String> getWords(Scanner input) {
        // read all words and sort
        Set<String> words = new TreeSet<String>();
        while (input.hasNext()) {
            String next = input.next().toLowerCase();
            words.add(next);
        }
        return result;
    }
    // post: explains program to user
    public static void giveIntro() {
        System.out.println("This program compares the vocabulary of two");
        System.out.println("text files, reporting the number of words");
        System.out.println("in common and the percent of overlap.");
        System.out.println();
    }
    // pre : overlap contains the words in commmon between list1 and list2
    // post: reports statistics about two word lists and their overlap
    public static void reportResults(Set<String> set1,
            Set<String> set2, Set<String> overlap) {
        System.out.println("file #1 words = " + set1.size());
        System.out.println("file #2 words = " + set2.size());
        System.out.println("common words  = " + overlap.size());
        double percent1 = 100.0 * overlap.size() / set1.size();
        double percent2 = 100.0 * overlap.size() / set2.size();
        System.out.println("% of file 1 in overlap = " + percent1);
        System.out.println("% of file 2 in overlap = " + percent2);
    }
}
```

Chapter 12

1.

```
public String starString(int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    } else if (n == 0) {
        return "*";
    } else {
        return starString(n - 1) + starString(n - 1);
    }
}
```

2.

```
public void writeNums(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    } else if (n == 1) {
        System.out.print(1);
    } else {
        writeNums(n - 1);
        System.out.print(", " + n);
    }
}
```

3.

```
public void writeSequence(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    } else if (n == 1) {
        System.out.print(1);
    } else if (n == 2) {
        System.out.print("1 1");
    } else {
        int number = (n + 1) / 2;
        System.out.print(number + " ");
        writeSequence(n - 2);
        System.out.print(" " + number);
    }
}
```

4.

```
public static int doubleDigits(int n) {
    if (n < 0) {
        return -doubleDigits(-n);
    } else if (n == 0) {
        return 0;
    } else {
        int digit = n % 10;
        int rest = n / 10;
        return digit + 10 * digit + 100 * doubleDigits(rest);
    }
}
```

5.

```
public static void writeBinary(int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    }
    if (n >= 2) {
        writeBinary(n / 2);
    }
    System.out.print(n % 2);
}
```

6.

```
public void writeSquares(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    } else if (n == 1) {
        System.out.print(1);
    } else if (n % 2 == 1) {
        System.out.print(n * n + ", ");
        writeSquares(n - 1);
    } else {
        writeSquares(n - 1);
        System.out.print(", " + n * n);
    }
}
```

7.

```
public void writeChars(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    } else if (n == 1) {
        System.out.print("*");
    } else if (n == 2) {
        System.out.print("**");
    } else {
        System.out.print("<");
        writeChars(n - 2);
        System.out.print(">");
    }
}
```

8.

```
public int multiplyEvens(int n) {
    if (n < 1) {
        throw new IllegalArgumentException();
    } else if (n == 1) {
        return 2;
    } else {
        return 2 * n * multiplyEvens(n - 1);
    }
}
```

9.

```
public double sumTo(int n) {
    if (n < 0) {
        throw new IllegalArgumentException();
    } else if (n == 0) {
        return 0.0;
    } else {
        return sumTo(n - 1) + 1.0 / n;
    }
}
```

10.

```
public static int digitMatch(int x, int y) {
    if (x < 0 || y < 0) {
        throw new IllegalArgumentException();
    } else if (x < 10 || y < 10) {
        if (x % 10 == y % 10) {
            return 1;
        } else {
            return 0;
        }
    } else if (x % 10 == y % 10) {
        return 1 + digitMatch(x / 10, y / 10);
    } else {
        return digitMatch(x / 10, y / 10);
    }
}
```

11.

```
public static String repeat(String s, int n) {
    if(n < 0) {
        throw new IllegalArgumentException();
    } else if(n == 0) {
        return "";
    } else if (n == 1) {
        return s;
    } else if (n % 2 == 0) {
        String temp = repeat(s, n / 2);
        return temp + temp;
    } else {
        return s + repeat(s, n - 1);
    }
}
```

```
public static String repeat(String s, int n) {
    if(n < 0) {
        throw new IllegalArgumentException();
    } else if(n == 0) {
        return "";
    } else if (n % 2 == 0) {
        return repeat(s + s, n / 2);
    } else {
        return s + repeat(s, n - 1);
    }
}
```

12.

```
public static boolean isReverse(String s1, String s2) {
    if (s1.length() == 0 && s2.length() == 0) {
        return true;
    } else if (s1.length() == 0 || s2.length() == 0) {
        return false;  // not same length
    } else {
        String s1first = s1.substring(0, 1);
        String s2last = s2.substring(s2.length() - 1);
        return s1first.equalsIgnoreCase(s2last) &&
                isReverse(s1.substring(1), s2.substring(0, s2.length() - 1));
    }
}
```

```
public static boolean isReverse(String s1, String s2) {
    if (s1.length() != s2.length()) {
        return false;  // not same length
    } else if (s1.length() == 0 && s2.length() == 0) {
        return true;
    } else {
        s1 = s1.toLowerCase();
        s2 = s2.toLowerCase();
        return s1.charAt(0) == s2.charAt(s2.length() - 1) &&
                isReverse(s1.substring(1, s1.length()),
                          s2.substring(0, s2.length() - 1));
    }
}
```

```
public static boolean isReverse(String s1, String s2) {
    if (s1.length() == s2.length()) {
        return isReverse(s1.toLowerCase(), 0, s2.toLowerCase(), s2.length() - 1);
    } else {
        return false;   // not same length
    }
}
private static boolean isReverse(String s1, int i1, String s2, int i2) {
    if (i1 >= s1.length() && i2 < 0) {
        return true;
    } else {
        return s1.charAt(i1) == s2.charAt(i2) &&
               isReverse(s1, i1 + 1, s2, i2 - 1);
    }
}
```

```
public static boolean isReverse(String s1, String s2) {
    return reverse(s1.toLowerCase()).equals(s2.toLowerCase());
}
private static String reverse(String s) {
    if (s.length() == 0) {
        return s;
    } else {
        return reverse(s.substring(1)) + s.charAt(0);
    }
}
```

13.

```
public static int indexOf(String source, String target) {
    if(target.length() > source.length()) {
        return -1;
    } else if(source.substring(0, target.length()).equals(target)) {
        return 0;
    } else {
        int pos = indexOf(source.substring(1), target);
        if(pos == -1) {
            return -1;
        } else {
            return pos + 1;
        }
    }
}
```

14.

```
public int evenDigits(int n) {
         if (n < 0) {
                  return -evenDigits(-n);
         } else if (n == 0) {
                  return 0;
         } else if (n % 2 == 0) {
                  return 10 * evenDigits(n / 10) + n % 10;
         } else {
                  return evenDigits(n / 10);
         }
}
```

15.

```
public static int permut(int n, int r) {
    if (r == 0) {
        return 1;
    } else {
        return permut(n - 1, r - 1) * n;
    }
}
```

16.

```
// Draws the Sierpinski Carpet fractal image.
import java.awt.*;
import java.util.*;
public class SierpinskiCarpet {
    public static final int SIZE = 243;
    public static void main(String[] args) {
        // prompt for level
        Scanner console = new Scanner(System.in);
        System.out.print("What level do you want? ");
        int level = console.nextInt();
        // initialize drawing panel
        DrawingPanel p = new DrawingPanel(SIZE, SIZE);
        Graphics g = p.getGraphics();
        drawFigure(g, level, 0, 0, SIZE);
    }
    // Draws a Sierpinski carpet to the given level in the given area.
    public static void drawFigure(Graphics g, int level,
                                  int x, int y, int size) {
        if (level > 0) {
            int size2 = size / 3;
            g.fillRect(x + size2, y + size2, size2, size2);

            // recursive calls
            for (int row = 0; row < 3; row++) {
                for (int col = 0; col < 3; col++) {
                    drawFigure(g, level - 1, x + col * size2,
                               y + row * size2, size2);
                }
            }
        }
    }
}
```

17.

```
// Draws the Cantor Set fractal image.
import java.awt.*;
import java.util.*;
public class CantorSet {
    public static final int SIZE = 243;
    public static void main(String[] args) {
        // prompt for level
        Scanner console = new Scanner(System.in);
        System.out.print("What level do you want? ");
        int level = console.nextInt();
        // initialize drawing panel
        DrawingPanel p = new DrawingPanel(SIZE, SIZE);
        Graphics g = p.getGraphics();
        drawFigure(g, level, 0, SIZE / 2, SIZE);
    }
    // Draws a Cantor Set to the given level in the given area.
    public static void drawFigure(Graphics g, int level,
                                  int x, int y, int size) {
        if (level > 0) {
            g.drawLine(x, y, x + size, y);
            drawFigure(g, level - 1, x, y + 2, size / 3);
            drawFigure(g, level - 1, x + 2 * size / 3, y + 2, size / 3);
        }
    }
}
```

18.

```
// Outputs all ways to climb the given number of stairs, taking
// 1 or 2 stairs with each step.
// Precondition: stairs >= 0
public static void waysToClimb(int stairs) {
    Stack<Integer> chosen = new Stack<Integer>();
    waysToClimb(stairs, chosen);
}

// Outputs all ways to climb the given number of stairs, taking
// 1 or 2 stairs with each step, with the given previously chosen steps.
private static void waysToClimb(int stairs, Stack<Integer> chosen) {
    if (stairs <= 0) {
        System.out.println(chosen);
    } else {
        chosen.push(1);                     // choose 1
        waysToClimb(stairs - 1, chosen);    // explore
        chosen.pop();                       // un-choose

        if (stairs > 1) {
            chosen.push(2);                  // choose 2
            waysToClimb(stairs - 2, chosen); // explore
            chosen.pop();                    // un-choose
        }
    }
}
```

19.

```
public static void countBinary(int n) {
    countBinary(n, "");
}

private static void countBinary(int digitsLeft, String s) {
    if (digitsLeft == 0) {
        System.out.println(s);
    } else {
        countBinary(digitsLeft - 1, s + "0");
        countBinary(digitsLeft - 1, s + "1");
    }
}
```

20.

```
// Prints all sub-lists of the given list of Strings.
// Precondition: elements != null and elements contains no duplicates
public static void subsets(List<String> elements) {
    List<String> chosen = new ArrayList<String>();
    explore(elements, chosen);
}

// Private recursive helper to explore all sub-lists of the given list of
// elements, assuming the given list of strings have already been chosen.
private static void explore(List<String> elements, List<String> chosen) {
    if (elements.isEmpty()) {
        System.out.println(chosen);   // base case; nothing left to choose
    } else {
        String first = elements.remove(0);   // make a choice: 1st element

        // two explorations: one with this first element, one without
        chosen.add(first);
        explore(elements, chosen);
        chosen.remove(chosen.size() - 1);
        explore(elements, chosen);

        elements.add(0, first);     // backtrack!  put 1st element back
    }
}
```

```
public static void subsets(List<String> elements) {
    List<String> chosen = new ArrayList<String>();
    subsets(elements, chosen, 0);
}

private static void subsets(List<String> elements,
                            List<String> chosen, int index) {
    if (index == elements.size()) {
        System.out.println(chosen);
    } else {
        String choice = elements.get(index);
        chosen.add(choice);
        subsets(elements, chosen, index + 1);
        chosen.remove(chosen.size() - 1);
        subsets(elements, chosen, index + 1);
    }
}
```

21.

```
public static int maxSum(List<Integer> numbers, int limit) {
    if (limit <= 0 || numbers.isEmpty()) {
        return 0;
    } else {
        int first = numbers.get(0);
        numbers.remove(0);

        int max;
        if (first > limit) {
            max = maxSum(numbers, limit);
        } else {
            int with    = first + maxSum(numbers, limit - first);
            int without = maxSum(numbers, limit);
            max = Math.max(with, without);
        }

        numbers.add(0, first);
        return max;
    }
}
```

22.

```
// Prints all ways to express n as a sum of squares of unique integers.
// Precondition: n >= 0
public static void printSquares(int n) {
    Set<Integer> chosen = new TreeSet<Integer>();
    explore(n, 1, chosen);
}

// Explore all ways to form n as a sum of squares of integers starting
// with the given min and storing the chosen results in the given set.
private static void explore(int n, int min, Set<Integer> chosen) {
    if (n == 0) {
        printHelper(chosen);   // base case: sum has reached n
    } else {
        // recursive case: try all combinations of every integer
        int max = (int) Math.sqrt(n);   // valid choices go up to sqrt(n)

        for (int i = min; i <= max; i++) {
            // try all combinations that include the square of this integer
            chosen.add(i);
            explore(n - (i * i), i + 1, chosen);
            chosen.remove(i);                        // backtrack
        }
    }
}
```

Chapter 13

1.
    a. examines indexes 4, 7, 8, 9; returns 9
    b. examines indexes 4, 7, 5, 6; returns -7
    c. examines indexes 4, 1; returns 1
    d. examines indexes 4, 1, 0; returns -2

2.
    a. examines indexes 5, 2, 3, 4; returns -5
    b. examines indexes 5, 8, 6, 7; returns 7
    c. examines indexes 5, 8, 10; returns 10
    d. examines indexes 5, 2, 0, 1; returns -2

3.
    a. examines indexes 7, 11, 9, 8; returns 8
    b. examines indexes 7, 3, 1, 2; returns -4
    c. examines indexes 7, 11, 13, 12; returns 12
    d. examines indexes 7, 11, 13, 14; returns -15

4. O(N)

5. O(N)

6. O(N^2)

7. O(N)

8. selection sort:

```
after pass 1      {9, 63, 45, 72, 27, 18, 54, 36}
after pass 2      {9, 18, 45, 72, 27, 63, 54, 36}
after pass 3      {9, 18, 27, 72, 45, 63, 54, 36}
after pass 4      {9, 18, 27, 36, 45, 63, 54, 72}
after pass 5      {9, 18, 27, 36, 45, 63, 54, 72}
after pass 6      {9, 18, 27, 36, 45, 54, 63, 72}
after pass 7      {9, 18, 27, 36, 45, 54, 63, 72}

after pass 1      {12, 29, 19, 48, 23, 55, 74, 37}
after pass 2      {12, 19, 29, 48, 23, 55, 74, 37}
after pass 3      {12, 19, 23, 48, 29, 55, 74, 37}
after pass 4      {12, 19, 23, 29, 48, 55, 74, 37}
after pass 5      {12, 19, 23, 29, 37, 55, 74, 48}
after pass 6      {12, 19, 23, 29, 37, 48, 74, 55}
after pass 7      {12, 19, 23, 29, 37, 48, 55, 74}
```

9. merge sort:

```
1st split         {63, 9, 45, 72} {27, 18, 54, 36}
2nd split         {63, 9} {45, 72} {27, 18} {54, 36}
3rd split         {63} {9} {45} {72} {27} {18} {54} {36}
1st merge         {9, 63} {45, 72} {18, 27} {36, 54}
2nd merge         {9, 45, 63, 72} {18, 27, 36, 54}
3rd merge         {9, 18, 27, 36, 45, 54, 63, 72}

1st split         {37, 29, 19, 48} {23, 55, 74, 12}
2nd split         {37, 29} {19, 48} {23, 55} {74, 12}
3rd split         {37} {29} {19} {48} {23} {55} {74} {12}
1st merge         {29, 37} {19, 48} {23, 55} {12, 74}
2nd merge         {19, 29, 37, 48} {12, 23, 55, 74}
3rd merge         {12, 19, 23, 29, 37, 48, 55, 74}
```

10. selection sort:

```
after pass 1      {-9, 5, 8, 14, 0, -1, -7, 3}
after pass 2      {-9, -7, 8, 14, 0, -1, 5, 3}
after pass 3      {-9, -7, -1, 14, 0, 8, 5, 3}
after pass 4      {-9, -7, -1, 0, 14, 8, 5, 3}
after pass 5      {-9, -7, -1, 0, 3, 8, 5, 14}
after pass 6      {-9, -7, -1, 0, 3, 5, 8, 14}
after pass 7      {-9, -7, -1, 0, 3, 5, 8, 14}

after pass 1      {-4, 56, 24, 5, 39, 15, 27, 10}
after pass 2      {-4, 5, 24, 56, 39, 15, 27, 10}
after pass 3      {-4, 5, 10, 56, 39, 15, 27, 24}
after pass 4      {-4, 5, 10, 15, 39, 56, 27, 24}
after pass 5      {-4, 5, 10, 15, 24, 56, 27, 39}
after pass 6      {-4, 5, 10, 15, 24, 27, 56, 39}
after pass 7      {-4, 5, 10, 15, 24, 27, 39, 56}
```

11. merge sort:

```
1st split         {8, 5, -9, 14} {0, -1, -7, 3}
2nd split         {8, 5} {-9, 14} {0, -1} {-7, 3}
3rd split         {8} {5} {-9} {14} {0} {-1} {-7} {3}
1st merge         {5, 8} {-9, 14} {-1, 0} {-7, 3}
2nd merge         {-9, 5, 8, 14} {-7, -1, 0, 3}
3rd merge         {-9, -7, -1, 0, 3, 5, 8, 14}

1st split         {15, 56, 24, 5} {39, -4, 27, 10}
2nd split         {15, 56} {24, 5} {39, -4} {27, 10}
3rd split         {15} {56} {24} {5} {39} {-4} {27} {10}
1st merge         {15, 56} {5, 24} {-4, 39} {10, 27}
2nd merge         {5, 15, 24, 56} {-4, 10, 27, 39}
3rd merge         {-4, 5, 10, 15, 24, 27, 39, 56}
```

12. selection sort:

```
after pass 1      {11, 44, 22, 88, 66, 33, 55, 77}
after pass 2      {11, 22, 44, 88, 66, 33, 55, 77}
after pass 3      {11, 22, 33, 88, 66, 44, 55, 77}
after pass 4      {11, 22, 33, 44, 66, 88, 55, 77}
after pass 5      {11, 22, 33, 44, 55, 88, 66, 77}
after pass 6      {11, 22, 33, 44, 55, 66, 88, 77}
after pass 7      {11, 22, 33, 44, 55, 66, 77, 88}

after pass 1      {-7, -6, -1, -5, 0, -2, -4, -3}
after pass 2      {-7, -6, -1, -5, 0, -2, -4, -3}
after pass 3      {-7, -6, -5, -1, 0, -2, -4, -3}
after pass 4      {-7, -6, -5, -4, 0, -2, -1, -3}
after pass 5      {-7, -6, -5, -4, -3, -2, -1, 0}
after pass 6      {-7, -6, -5, -4, -3, -2, -1, 0}
after pass 7      {-7, -6, -5, -4, -3, -2, -1, 0}
```

13. merge sort:

```
1st split         {22, 44, 11, 88} {66, 33, 55, 77}
2nd split         {22, 44} {11, 88} {66, 33} {55, 77}
3rd split         {22} {44} {11} {88} {66} {33} {55} {77}
1st merge         {22, 44} {11, 88} {33, 66} {55, 77}
2nd merge         {11, 22, 44, 88} {33, 55, 66, 77}
3rd merge         {11, 22, 33, 44, 55, 66, 77, 88}

1st split         {-3, -6, -1, -5} {0, -2, -4, -7}
2nd split         {-3, -6} {-1, -5} {0, -2} {-4, -7}
3rd split         {-3} {-6} {-1} {-5} {0} {-2} {-4} {-7}
1st merge         {-6, -3} {-5, -1} {-2, 0} {-7, -4}
2nd merge         {-6, -5, -3, -1} {-7, -4, -2, 0}
3rd merge         {-7, -6, -5, -4, -3, -2, -1, 0}
```

14.

```
// Searches for a word in a dictionary text file
// and reports that word's position in the file.
import java.io.*;
import java.util.*;
public class WordsBetween {
    public static void main(String[] args) throws FileNotFoundException {
        // read sorted dictionary file into an ArrayList
        Scanner in = new Scanner(new File("words.txt"));
        ArrayList<String> words = new ArrayList<String>();
        while (in.hasNext()) {
            String word = in.next();
            words.add(word);
        }

        // binary search the list for a particular word
        System.out.print("Type two words: ");
        Scanner console = new Scanner(System.in);
        String word1 = console.next();
        String word2 = console.next();

        int index1 = Collections.binarySearch(words, word1);
        int index2 = Collections.binarySearch(words, word2);
        if (index1 >= 0 && index2 >= 0) {
            int between = Math.min(Math.abs(index2 - index1) - 1, 0);
            System.out.println("There are " + between + " words between "
                               + word1 + " and " + word2);
        } else {
            System.out.println(word1 + " or " + word2 + " is not found");
        }
    }
}
```

15.

```
// Compares Points by their distance from the origin.
import java.awt.*;
import java.util.*;
public class PointDistanceComparator implements Comparator<Point> {
    public int compare(Point p1, Point p2) {
        double d1 = p1.distance(0, 0);
        double d2 = p2.distance(0, 0);

        if (d1 > d2) {
            return 1;
        } else if (d1 < d2) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

16.

```
// Compares Strings by their number of words.
import java.util.*;
public class StringWordComparator implements Comparator<String> {
    public int compare(String s1, String s2) {
        int count1 = wordCount(s1);
        int count2 = wordCount(s2);

        if (count1 > count2) {
            return 1;
        } else if (count1 < count2) {
            return -1;
        } else {
            return 0;
        }
    }
    // wordCount algorithm from Chapter 4 exercises
    public int wordCount(String s) {
        int count = 0;
        if (s.charAt(0) != ' ')
            count++;
        for (int i = 0; i < s.length() - 1; i++ ) {
            if (s.charAt(i) == ' ' && s.charAt(i + 1) != ' ') {
                count++;
            }
        }
        return count;
    }
}
```

17.

```
// Compares Strings in a particular format.
// Example: "123456 Seattle, WA", beginning with a numeric token that is followed by additional text tokens
// Example, "276453 Helena, MT" is greater than "9847 New York, NY".
import java.util.*;
public class CityComparator implements Comparator<String> {
    public int compare(String s1, String s2) {
        Scanner scan1 = new Scanner(s1);
        Scanner scan2 = new Scanner(s2);
        int code1 = scan1.nextInt();
        int code2 = scan2.nextInt();
        if (code1 != code2) {
            return code1 - code2;
        }

        String city1 = scan1.next().replace(",", "");
        String city2 = scan2.next().replace(",", "");
        if (!city1.equals(city2)) {
            return city1.compareTo(city2);
        }

        String state1 = scan1.next();
        String state2 = scan2.next();
        return state1.compareTo(state2);
    }
}
```

18.

```
// Places the elements of the given array into sorted order
// using the selection sort algorithm.
// post: array is in sorted (nondecreasing) order
public static void selectionSortEnd(int[] a) {
    for (int i = 0; i < a.length; i++) {
        // find index of largest element
        int largest = 0;
        for (int j = 0; j < a.length - i; j++) {
            if (a[j] > a[largest]) {
                largest = j;
            }
        }

        int temp = a[length - 1 - i];       // swap largest to end
        a[length - 1 - i] = a[largest];
        a[largest] = temp;
    }
}
```

19.

```
// Places the elements of the given array into sorted order
// using the selection sort algorithm.
// post: array is in sorted (nondecreasing) order
public static void dualSelectionSort(int[] a) {
    for (int i = 0; i < a.length / 2; i++) {
        // find index of smallest/largest element
        int smallest = i;
        int largest = i;
        for (int j = i + 1; j < a.length - i; j++) {
            if (a[j] < a[smallest]) {
                smallest = j;
            } else if (a[j] > a[largest]) {
                largest = j;
            }
        }

        int temp = a[i];       // swap smallest to front
        a[i] = a[smallest];
        a[smallest] = temp;
        if (largest == i) {
            largest = smallest;
        }

        int temp2 = a[length - 1 - i];       // swap largest to end
        a[length - 1 - i] = a[largest];
        a[largest] = temp2;
    }
}
```

20.

```
public static void shuffle(int[] a) {
    Random rand = new Random();
    for (int i = 0; i < a.length - 1; i++) {
        int j = rand.nextInt(a.length - i) + i;

        // swap elements i and j
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

21.

```
// Places the elements of a into sorted order.
public static void bogoSort(int[] a) {
    while (!isSorted(a)) {
        shuffle(a);
    }
}
// Returns true if a's elements are in sorted order.
public static boolean isSorted(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i + 1]) {
            return false;
        }
    }
    return true;
}
// Shuffles an array of ints by randomly swapping each
// element with an element ahead of it in the array.
public static void shuffle(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        // pick a random index in [i+1, a.length-1]
        int range = a.length - 1 - (i + 1) + 1;
        int j = (int) (Math.random() * range + (i + 1));
        swap(a, i, j);
    }
}
// Swaps a[i] with a[j].
public static void swap(int[] a, int i, int j) {
    if (i != j) {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
```

Chapter 14

1.

```
public void splitStack(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();

    // transfer all elements from stack to queue
    int oldLength = s.size();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }

    // transfer negatives from queue to stack
    for (int i = 1; i <= oldLength; i++) {
        int n = q.remove();
        if (n < 0) {
            s.push(n);
        } else {
            q.add(n);
        }
    }

    // transfer nonnegatives from queue to stack
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

2.

```
public void stutter(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while(!q.isEmpty()) {
        int n = q.remove();
        s.push(n);
        s.push(n);
    }
}
```

3.

```
public Stack<Integer> copyStack(Stack<Integer> s) {
    Stack<Integer> s2 = new Stack<Integer>();
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        s2.push(s.pop());
    }
    while(!s2.isEmpty()) {
        q.add(s2.pop());
    }
    while (!q.isEmpty()) {
        int n = q.remove();
        s.push(n);
        s2.push(n);
    }
    return s2;
}
```

4.

```
public static void collapse(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (q.size() > 1) {
        s.push(q.remove() + q.remove());
    }
    if (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

5.

```
public boolean equals(Stack<Integer> s1, Stack<Integer> s2) {
    Stack<Integer> s3 = new Stack<Integer>();
    boolean same = true;
    while (same && !s1.isEmpty() && !s2.isEmpty()) {
        int num1 = s1.pop();
        int num2 = s2.pop();
        if (num1 != num2) {
            same = false;
        }
        s3.push(num1);
        s3.push(num2);
    }
    same = same && s1.isEmpty() && s2.isEmpty();
    while (!s3.isEmpty()) {
        s2.push(s3.pop());
        s1.push(s3.pop());
    }
    return same;
}
```

6.

```
public void rearrange(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();
    int oldSize = q.size();
    for (int i = 0; i < oldSize; i++) {
        int n = q.remove();
        if (n % 2 == 0) {
            q.add(n);
        } else {
            s.push(n);
        }
    }
    int evenCount = q.size();
    while (!s.isEmpty()){
        q.add(s.pop());
    }
    for (int i = 0; i < evenCount; i++) {
        q.add(q.remove());
    }
    for (int i = 0; i < oldSize - evenCount; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}
```

7.

```
public void reverseHalf(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();

    int oldSize = q.size();

    for(int i = 0; i < oldSize; i++) {
        if(i % 2 == 0) {
            q.add(q.remove());
        } else {
            s.push(q.remove());
        }
    }

    for(int i = 0; i < oldSize; i++) {
        if(i % 2 == 0) {
            q.add(q.remove());
        } else {
            q.add(s.pop());
        }
    }
}
```

8.

```
public boolean isPalindrome(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();
    for (int i = 0; i < q.size(); i++) {
        int n = q.remove();
        q.add(n);
        s.push(n);
    }
    boolean ok = true;
    for (int i = 0; i < q.size(); i++) {
        int n1 = q.remove();
        int n2 = s.pop();
        if (n1 != n2) {
            ok = false;
        }
        q.add(n1);
    }
    return ok;
}
```

9.

```
public void switchPairs(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (q.size() > 1) {
        int n1 = q.remove();
        int n2 = q.remove();
        s.push(n2);
        s.push(n1);
    }
    if (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

10.

```
public boolean isConsecutive(Stack<Integer> s) {
    if (s.size() <= 1) {
        return true;
    } else {
        Queue<Integer> q = new LinkedList<Integer>();
        int prev = s.pop();
        q.add(prev);
        boolean ok = true;
        while (!s.isEmpty()) {
            int next = s.pop();
            if (prev - next != 1) {
                ok = false;
            }
            q.add(next);
            prev = next;
        }
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        while (!s.isEmpty()) {
            q.add(s.pop());
        }
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        return ok;
    }
}
```

11.

```
public void reorder(Queue<Integer> q) {
    Stack<Integer> s = new Stack<Integer>();
    int oldSize = q.size();
    for (int i = 0; i < oldSize; i++) {
        int n = q.remove();
        if (n < 0) {
            s.push(n);
        } else {
            q.add(n);
        }
    }
    int newSize = q.size();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    for (int i = 0; i < newSize; i++) {
        q.add(q.remove());
    }
}
```

12.

```
public void shift(Stack<Integer> s, int n) {
    Queue<Integer> q = new LinkedList<Integer>();
    int otherSize = s.size() - n;
    for (int i = 0; i < otherSize; i++) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

13.

```
// using a temporary stack
public static void expunge(Stack<Integer> s) {
    if (!s.isEmpty()) {
        // copy sorted contents into temp stack s2
        Stack<Integer> s2 = new Stack<Integer>();
        int prev;
        while (!s.isEmpty()) {
            prev = s.pop();
            while (!s.isEmpty() && s.peek() < prev) {
                s.pop();
            }
            s2.push(prev);
        }

        // transfer s2 back into s
        while (!s2.isEmpty()) {
            s.push(s2.pop());
        }
    }
}
```

14.

```
public static void reverseFirstK(int k, Queue<Integer> q) {
    if (q == null || k > q.size()) {
        throw new IllegalArgumentException();
    } else if (k > 0) {
        Stack<Integer> s = new Stack<Integer>();   // first k elements -> S
        for (int i = 0; i < k; i++) {
            s.push(q.remove());
        }

        while (!s.isEmpty()) {                                 // s2q(s, q);
            q.add(s.pop());
        }

        for (int i = 0; i < q.size() - k; i++) {   // wrap around rest of elements so
            q.add(q.remove());                        // k reversed ones appear at front
        }
    }
}
```

15.

```
public static boolean isSorted(Stack<Integer> s) {
    if (s.size() < 2) {
        return true;
    }

    boolean sorted = true;
    int prev = s.pop();
    Stack<Integer> backup = new Stack<Integer>();
    backup.push(prev);
    while (!s.isEmpty()) {
        int curr = s.pop();
        backup.push(curr);
        if (prev > curr) {
            sorted = false;
        }
        prev = curr;
    }

    while (!backup.isEmpty()) {   // restore s
        s.push(backup.pop());
    }

    return sorted;
}
```

16.

```
public static void mirror(Stack<Integer> s) {
    if (s == null)
        throw new IllegalArgumentException();
    Queue<Integer> q = new LinkedList<Integer>();

    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    for (int i = 0; i < q.size(); i++) {
        int n = q.remove();
        q.add(n);
        s.push(n);
    }
    int size = q.size();
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
    for(int i = 0; i < size; i++) {
        q.add(q.remove());
    }
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

17.

```
public void compressDuplicates(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();
    while (!s.isEmpty()) {
        q.add(s.pop());       // s -> q
    }
    while (!q.isEmpty()) {
        s.push(q.remove());   // q -> s, to reverse the stack order
    }
    while (!s.isEmpty()) {     // s -> q
        q.add(s.pop());
    }
    if (!q.isEmpty()) {       // q -> s, replacing dupes with (count, val)
        int last = q.remove();
        int count = 1;
        while (!q.isEmpty()) {
            int next = q.remove();
            if (next == last) {
                count++;
            } else {
                s.push(count);
                s.push(last);
                count = 1;
                last = next;
            }
        }
        s.push(count);
        s.push(last);
    }
}
```

18.

```
public static void mirrorHalves(Queue<Integer> q) {
    if (q == null || q.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }

    Stack<Integer> s = new Stack<Integer>();
    int size = q.size();

    for (int i = 0; i < size / 2; i++) {
        int element = q.remove();
        s.push(element);
        q.add(element);
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }

    for (int i = 0; i < size / 2; i++) {
        int element = q.remove();
        s.push(element);
        q.add(element);
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}
```

19.

```
public static int removeMin(Stack<Integer> s) {
    Queue<Integer> q = new LinkedList<Integer>();

    int min = s.pop();
    q.add(min);
    while(!s.isEmpty()) {
        int next = s.pop();
        if(next < min) {
            min = next;
        }
        q.add(next);
    }

    while(!q.isEmpty()) {
        int next = q.remove();
        if(next != min) {
            s.push(next);
        }
    }

    s2q(s, q);
    q2s(q, s);

    return min;
}
```

20.

```
public static void interleave(Queue<Integer> q) {
    if (q.size() % 2 != 0) {
        throw new IllegalArgumentException();
    }
    Stack<Integer> s = new Stack<Integer>();
    int halfSize = q.size() / 2;
    for (int i = 0; i < halfSize; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {   // s2q(s, q);
        q.add(s.pop());
    }
    for (int i = 0; i < halfSize; i++) {
        q.add(q.remove());
    }
    for (int i = 0; i < halfSize; i++) {
        s.push(q.remove());
    }
    while (!s.isEmpty()) {
        q.add(s.pop());
        q.add(q.remove());
    }
}
```

Chapter 15

1.

```
public int lastIndexOf(int value) {
    for (int i = size - 1; i >= 0; i--) {
        if (elementData[i] == value) {
            return i;
        }
    }
    return -1;
}
```

2.

```
public int indexOfSubList(ArrayIntList sublist) {
    for (int i = 0; i < size - sublist.size; i++) {
        boolean match = true;
        for (int j = 0; j < sublist.size && match; j++) {
            if (elementData[i + j] != sublist.elementData[j]) {
                match = false;
            }
        }
        if (match) {
            return i;
        }
    }
    return -1;
}
```

3.

```
public void replaceAll(int oldValue, int newValue) {
    for (int i = 0; i < size; i++) {
        if (elementData[i] == oldValue) {
            elementData[i] = newValue;
        }
    }
}
```

4.

```
public void reverse() {
    for (int i = 0; i < size / 2; i++) {
        int temp = elementData[i];
        elementData[i] = elementData[size - 1 - i];
        elementData[size - 1 - i] = temp;
    }
}
```

5.

```
public ArrayIntList runningTotal() {
    ArrayIntList result = new ArrayIntList(elementData.length);
    if (size > 0) {
        result.add(elementData[0]);
        for (int i = 1; i < size; i++) {
            result.add(result.get(i - 1) + elementData[i]);
        }
    }
    return result;
}
```

```
public ArrayIntList runningTotal() {
    ArrayIntList result = new ArrayIntList(elementData.length);
    if (size > 0) {
        result.elementData[0] = elementData[0];
        for (int i = 1; i < size; i++) {
            result.elementData[i] = result.elementData[i - 1] + elementData[i];
        }
        result.size = size;
    }
    return result;
}
```

6.

```
public void fill(int value) {
    for (int i = 0; i < size; i++) {
        elementData[i] = value;
    }
}
```

7.

```
public boolean isPairwiseSorted() {
    for (int i = 0; i < size - 1; i += 2) {
        if (elementData[i] > elementData[i + 1]) {
            return false;
        }
    }
    return true;
}
```

8.

```
public int count(int value) {
    int result = 0;
    for (int i = 0; i < size; i++) {
        if (elementData[i] == value) {
            result++;
        }
    }
    return result;
}
```

9.

```
public int maxCount() {
    if (size == 0) {
        return 0;
    } else {
        int max = 1;
        int count = 1;
        for (int i = 1; i < size; i++) {
            if (elementData[i] == elementData[i - 1]) {
                count++;
                if (count > max) {
                    max = count;
                } else {
                    count = 1;
                }
            }
        }
        return max;
    }
}
```

10.

```
public int longestSortedSequence() {
    if (size == 0) {
        return 0;
    }
    int max = 1;
    int current = 1;
    for (int i = 1; i < size; i++) {
        if (elementData[i] >= elementData[i - 1]) {
            current++;
            if (current > max) {
                max = current;
            }
        } else {
            current = 1;
        }
    }
    return max;
}
```

11.

```
public int removeLast() {
    if (size == 0) {
        throw new NoSuchElementException();
    }
    int value = elementData[size - 1];
    elementData[size - 1] = 0;   // optional
    size--;
    return value;
}
```

12.

```
public void removeFront(int n) {
    for (int i = n; i < size; i++) {
        elementData[i - n] = elementData[i];
    }
    size -= n;
}
```

13.

```
public void removeAll(int value) {
    int i = 0;
    while (i < size) {
        if (elementData[i] == value) {
            remove(i);
        } else {
            i++;
        }
    }
}
```

```
// This solution is much faster than the other because it is
// guaranteed to complete its work with one pass through the data.
public void removeAll(int value) {
    int newSize = 0;
    for (int i = 0; i < size; i++) {
        if (elementData[i] != value) {
            elementData[newSize] = elementData[i];
            newSize++;
        }
    }
    size = newSize;
}
```

14.

```
public void printInversions() {
    for (int i = 0; i < size - 1; i++) {
        for (int j = i + 1; j < size; j++) {
            if (elementData[i] > elementData[j]) {
                System.out.println("(" + elementData[i] + ", "
                                   + elementData[j] + ")");
            }
        }
    }
}
```

15.

```
public void mirror() {
    ensureCapacity(size * 2);
    int last = 2 * size - 1;
    for (int i = 0; i < size; i++) {
        elementData[last - i] = elementData[i];
    }
    size = size * 2;
}
```

```
public void mirror() {
    for (int i = size - 1; i >= 0; i--) {
        add(get(i));
    }
}
```

```
public void mirror() {
    ensureCapacity(size * 2);
    int i = 0, j = 2 * size - 1;
    while (i < j) {
        elementData[j--] = elementData[i++];
        size++;
    }
}
```

16.

```
public void stutter() {
    int newSize = 2 * size;
    ensureCapacity(newSize);
    for (int i = 0; i < newSize; i += 2) {
        add(i, elementData[i]);
    }
}
```

```
public void stutter() {
    ensureCapacity(size * 2);
    for (int i = 2 * size - 1; i > 0; i -= 2) {
        elementData[i] = elementData[i / 2];
        elementData[i - 1] = elementData[i / 2];
    }
}
```

17.

```
public void stretch(int n) {
    if (n > 0) {
        ensureCapacity(n * size);
        for (int i = size - 1; i >= 0; i--) {
            for (int j = 0; j < n; j++) {
                elementData[i * n + j] = elementData[i];
            }
        }
        size *= n;
    } else {
        size = 0;
    }
}
```

```
public void stretch(int n) {
    if (n > 1) {
        int finalSize = n * size;
        for (int i = 0; i < finalSize; i++) {
            if (i % n != 0) {
                add(i, get(i / n * n));
            }
        }
    } else if (n <= 0) {
        clear();
    }
}
```

```
public void stretch(int n) {
    if (n > 0) {
        for (int i = n * size - 1; i >= 0; i--) {
            if (i % n != 0) {
                add(i / n, get(i / n));
            }
        }
    } else {
        clear();
    }
}
```

18.

```
public void doubleList() {
    ensureCapacity(size * 2);
    for (int i = 0; i < size; i++) {
        elementData[i + size] = elementData[i];
    }
    size *= 2;
}
```

19.

```
public void compress() {
    for (int i = 0; i < size - 1; i += 2) {
        elementData[i / 2] = elementData[i] + elementData[i + 1];
    }
    if (size % 2 != 0) {
        elementData[size / 2] = elementData[size - 1];
    }
    size /= 2;
}
```

20.

```
public void rotate() {
    if (size > 0) {
        int first = elementData[0];
        for (int i = 1; i < size; i++) {
            elementData[i - 1] = elementData[i];
        }
        elementData[size - 1] = first;
    }
}
```

21.

```
public void switchPairs() {
    for (int i = 0; i < size - 1; i++) {
        int temp = elementData[i];
        elementData[i] = elementData[i + 1];
        elementData[i + 1] = temp;
    }
}
```

Chapter 16

1.

```
public void set(int index, int value) {
    ListNode current = front;
    for (int i = 0; i < index; i++) {
        current = current.next;
    }
    current.data = value;
}
```

2.

```
public int min() {
    if (front == null) {
        throw new NoSuchElementException("list is empty");
    } else {
        int min = front.data;
        ListNode current = front.next;
        while (current != null) {
            if (current.data < min) {
                min = current.data;
            }
            current = current.next;
        }
        return min;
    }
}
```

3.

```
public boolean isSorted() {
    if (front != null) {
        ListNode current = front;
        while (current.next != null) {
            if (current.data > current.next.data) {
                return false;
            }
            current = current.next;
        }
    }
    return true;
}
```

4.

```
public int lastIndexOf(int n) {
    int result = -1;
    int index = 0;
    ListNode current = this.front;
    while (current != null) {
        if (current.data == n) {
            result = index;
        }
        index++;
        current = current.next;
    }
    return result;
}
```

5.

```
public int countDuplicates() {
    int count = 0;
    if (front != null) {
        ListNode current = front;
        while (current.next != null) {
            if (current.data == current.next.data) {
                count++;
            }
            current = current.next;
        }
    }
    return count;
}
```

6.

```
public boolean hasTwoConsecutive() {
    if (front == null || front.next == null) {
        return false;
    }
    ListNode current = front;
    while (current.next != null) {
        if (current.data + 1 == current.next.data) {
            return true;
        }
        current = current.next;
    }
    return false;
}
```

7.

```
public int deleteBack() {
    if (front == null) {
        throw new NoSuchElementException("empty list");
    }
    int result = 0;
    if (front.next == null) {
        result = front.data;
        front = null;
    } else {
        ListNode current = front;
        while (current.next.next != null) {
            current = current.next;
        }
        result = current.next.data;
        current.next = null;
    }
    return result;
}
```

8.

```
public void switchPairs() {
    if (front != null && front.next != null) {
        ListNode current = front.next;
        front.next = current.next;
        current.next = front;
        front = current;
        current = current.next;

        while (current.next != null && current.next.next != null) {
            ListNode temp = current.next.next;
            current.next.next = temp.next;
            temp.next = current.next;
            current.next = temp;
            current = temp.next;
        }
    }
}
```

9.

```
public void stutter() {
    ListNode current = front;
    while (current != null) {
        current.next = new ListNode(current.data, current.next);
        current = current.next.next;
    }
}
```

10.

```
public void stretch(int factor) {
    if (factor <= 0) {
        front = null;
    } else {
        ListNode current = front;
        while (current != null) {
            current.data = current.data / factor;
            for (int i = 1; i < factor; i++) {
                current.next = new ListNode(current.data, current.next);
                current = current.next;
            }
            current = current.next;
        }
    }
}
```

11.

```
public void compress(int factor) {
    ListNode current = front;
    while (current != null) {
        int i = 1;
        ListNode current2 = current.next;
        while (current2 != null && i < factor) {
            current.data += current2.data;
            current.next = current.next.next;
            i++;
            current2 = current2.next;
        }
        current = current.next;
    }
}
```

12.

```
public void split() {
    if (front != null) {
        ListNode current = front;
        while (current.next != null) {
            if (current.next.data < 0) {
                ListNode temp = current.next;
                current.next = current.next.next;
                temp.next = front;
                front = temp;
            } else {
                current = current.next;
            }
        }
    }
}
```

13.

```
public void transferFrom(LinkedIntList other) {
    if (front == null) {
        front = other.front;
    } else {
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = other.front;
    }
    other.front = null;
}
```

14.

```
public void removeAll(int value) {
    while (front != null && front.data == value) {
        front = front.next;
    }
    if (front != null) {
        ListNode current = front;
        while (current.next != null) {
            if (current.next.data == value) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }
    }
}
```

15.

```
public boolean equals(LinkedIntList other) {
    ListNode current1 = front;
    ListNode current2 = other.front;
    while (current1 != null && current2 != null) {
        if (current1.data != current2.data) {
            return false;
        }
        current1 = current1.next;
        current2 = current2.next;
    }
    return current1 == null && current2 == null;
}
```

16.

```
public LinkedIntList removeEvens() {
    LinkedIntList result = new LinkedIntList();
    if (front != null) {
        result.front = front;
        front = front.next;
        ListNode current = front;
        ListNode resultLast = result.front;
        while (current != null && current.next != null) {
            resultLast.next = current.next;
            resultLast = current.next;
            current.next = current.next.next;
            current = current.next;
        }
        resultLast.next = null;
    }
    return result;
}
```

17.

```
public void removeRange(int low, int high) {
    if (low < 0 || high < 0) {
        throw new IllegalArgumentException();
    }
    if (low == 0) {
        while (high >= 0) {
            front = front.next;
            high--;
        }
    } else {
        ListNode current = front;
        int count = 1;
        while (count < low) {
            current = current.next;
            count++;
        }
        ListNode current2 = current.next;
        while (count < high) {
            current2 = current2.next;
            count++;
        }
        current.next = current2.next;
    }
}
```

18.

```
public void doubleList() {
    if (front != null) {
        ListNode half2 = new ListNode(front.data);
        ListNode back = half2;
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
            back.next = new ListNode(current.data);
            back = back.next;
        }
        current.next = half2;
    }
}
```

19.

```
public void rotate() {
    if (front != null && front.next != null) {
        ListNode temp = front;
        front = front.next;
        ListNode current = front;
        while (current.next != null) {
            current = current.next;
        }
        current.next = temp;
        temp.next = null;
    }
}
```

20.

```
public void shift() {
    if (front != null && front.next != null) {
        ListNode otherFront = front.next;
        front.next = front.next.next;
        ListNode current1 = front;
        ListNode current2 = otherFront;
        while (current1.next != null) {
            current1 = current1.next;
            if (current1.next != null) {
                current2.next = current1.next;
                current1.next = current1.next.next;
                current2 = current2.next;
            }
        }
        current2.next = null;
        current1.next = otherFront;
    }
}
```

21.

```
public void reverse() {
    ListNode current = front;
    ListNode previous = null;
    while (current != null) {
        ListNode nextNode = current.next;
        current.next = previous;
        previous = current;
        current = nextNode;
    }
    front = previous;
}
```

Chapter 17

1.

```
public int countLeftNodes() {
    return countLeftNodes(overallRoot);
}
private int countLeftNodes(IntTreeNode root) {
    if (root == null) {
        return 0;
    } else if (root.left == null) {
        return countLeftNodes(root.right);
    } else {
        return 1 + countLeftNodes(root.left) + countLeftNodes(root.right);
    }
}
```

2.

```
public int countEmpty() {
    return countEmpty(overallRoot);
}
private int countEmpty(IntTreeNode root) {
    if (root == null) {
        return 1;
    } else {
        return countEmpty(root.left) + countEmpty(root.right);
    }
}
```

3.

```
public int depthSum() {
    return depthSum(overallRoot, 1);
}
private int depthSum(IntTreeNode root, int depth) {
    if (root == null) {
        return 0;
    } else {
        return depth * root.data + depthSum(root.left, depth + 1)
                                 + depthSum(root.right, depth + 1);
    }
}
```

4.

```
public int countEvenBranches() {
    return countEvenBranches(overallRoot);
}
private int countEvenBranches(IntTreeNode root) {
    if (root == null) {
        return 0;
    } else if (root.left == null && root.right == null) {
        return 0;
    } else if (root.data % 2 == 0) {
        return 1 + countEvenBranches(root.left) + countEvenBranches(root.right);
    } else {
        return countEvenBranches(root.left) + countEvenBranches(root.right);
    }
}
```

```
public int countEvenBranches() {
    return countEvenBranches(overallRoot);
}
private int countEvenBranches(IntTreeNode root) {
    if(root == null || (root.left == null && root.right == null)) {
        return 0;
    } else {
        int result = 0;
        if (root.data % 2 == 0) {
            result = 1;
        }
        return result + countEvenBranches(root.left)
                      + countEvenBranches(root.right);
    }
}
```

5.

```
public void printLevel(int target) {
    if(target < 1) {
        throw new IllegalArgumentException();
    }
    printLevel(overallRoot, target, 1);
}
private void printLevel(IntTreeNode root, int target, int level) {
    if(root != null) {
        if(level == target) {
            System.out.println(root.data);
        } else {
            printLevel(root.left, target, level + 1);
            printLevel(root.right, target, level + 1);
        }
    }
}
```

```
public void printLevel(int target) {
    if (target < 1) {
        throw new IllegalArgumentException();
    }
    printLevel(overallRoot, target);
}
private void printLevel(IntTreeNode root, int target) {
    if (root != null) {
        if (target == 1) {
            System.out.println(root.data);
        } else {
            printLevel(root.left,  target - 1);
            printLevel(root.right, target - 1);
        }
    }
}
```

6.

```
public void printLeaves() {
    if (overallRoot == null) {
        System.out.println("no leaves");
    } else {
        System.out.print("leaves:");
        printLeaves(overallRoot);
        System.out.println();
    }
}
private void printLeaves(IntTreeNode root) {
    if (root != null) {
        if (root.left == null && root.right == null) {
            System.out.print(" " + root.data);
        } else {
            printLeaves(root.right);
            printLeaves(root.left);
        }
    }
}
```

7.

```
public boolean isFull() {
    return (overallRoot == null || isFull(overallRoot));
}
private boolean isFull(IntTreeNode root) {
    if(root.left == null && root.right == null) {
        return true;
    } else {
        return (root.left != null && root.right != null &&
            isFull(root.left) && isFull(root.right));
    }
}
```

```
public boolean isFull() {
    return (overallRoot == null || isFull(overallRoot));
}
private boolean isFull(IntTreeNode root) {
    if(root == null) {
        return false;
    } else if(root.left == null && root.right == null) {
        return true;
    } else {
        return (isFull(root.left) && isFull(root.right));
    }
}
```

```
public boolean isFull() {
    return isFull(overallRoot);
}
private boolean isFull(IntTreeNode root) {
    if (root == null || (root.left == null && root.right == null)) {
        return true;
    } else if (root.left == null || root.right == null) {
        return false;
    } else {
        return isFull(root.left) && isFull(root.right);
    }
}
```

8.

```
public String toString() {
    return toString(overallRoot);
}
private String toString(IntTreeNode root) {
    if (root == null) {
        return "empty";
    } else if (root.left == null && root.right == null) {
        return "" + root.data;
    } else {
        return "(" + root.data + ", " + toString(root.left) +
                ", " + toString(root.right) + ")";
    }
}
```

9.

```
public boolean equals(IntTree other) {
    return equals(this.overallRoot, other.overallRoot);
}
private boolean equals(IntTreeNode root1, IntTreeNode root2) {
    if (root1 == null || root2 == null) {
        return root1 == null && root2 == null;
    } else {
        return root1.data == root2.data
                && equals(root1.left, root2.left)
                && equals(root1.right, root2.right);
    }
}
```

10.

```
public void doublePositives() {
    doublePositives(overallRoot);
}
private void doublePositives(IntTreeNode root) {
    if (root != null) {
        if (root.data > 0) {
            root.data *= 2;
        }
        doublePositives(root.left);
        doublePositives(root.right);
    }
}
```

11.

```
public int numberNodes() {
    return numberNodes(overallRoot, 1);
}
private int numberNodes(IntTree.IntTreeNode root, int count) {
    if (root == null) {
        return 0;
    } else {
        root.data = count;
        int leftNum  = numberNodes(root.left,  count + 1);
        int rightNum = numberNodes(root.right, count + 1 + leftNum);
        return leftNum + rightNum + 1;
    }
}
```

```
public int numberNodes() {
    return numberNodes(overallRoot, 1);
}
private int numberNodes(IntTree.IntTreeNode root, int count) {
    if (root == null) {
        return count - 1;
    } else {
        root.data = count;
        count = numberNodes(root.left,  count + 1);
        count = numberNodes(root.right, count + 1);
        return count;
    }
}
```

12.

```
public void removeLeaves() {
    overallRoot = removeLeaves(overallRoot);
}
private IntTreeNode removeLeaves(IntTreeNode root) {
    if(root != null) {
        if(root.left == null && root.right == null) {
            root = null;
        } else {
            root.left  = removeLeaves(root.left);
            root.right = removeLeaves(root.right);
        }
    }
    return root;
}
```

13.

```
public IntTree copy() {
    IntTree result = new IntTree();
    result.overallRoot = copy(overallRoot);
    return result;
}
private IntTreeNode copy(IntTreeNode root) {
    if (root == null) {
        return null;
    } else {
        return new IntTreeNode(root.data, copy(root.left), copy(root.right));
    }
}
```

-
  ```
  public void completeToLevel(int target) {
      if (target < 1) {
          throw new IllegalArgumentException();
      }
      overallRoot = complete(overallRoot, target, 1);
  }
  private IntTreeNode complete(IntTreeNode root, int target, int level) {
      if (level <= target) {
          if (root == null) {
              root = new IntTreeNode(-1);
          }
          root.left  = complete(root.left,  target, level + 1);
          root.right = complete(root.right, target, level + 1);
      }
      return root;
  }
  ```

-
  ```
  public void completeToLevel(int target) {
      if (target < 1) {
          throw new IllegalArgumentException();
      }
      overallRoot = complete(overallRoot, target);
  }
  private IntTreeNode complete(IntTreeNode root, int target) {
      if (target > 0) {
          if (root == null) {
              root = new IntTreeNode(-1);
          }
          root.left  = complete(root.left,  target - 1);
          root.right = complete(root.right, target - 1);
      }
      return root;
  }
  ```

-
  ```
  public void trim(int min, int max) {
      overallRoot = trim(overallRoot, min, max);
  }
  private IntTreeNode trim(IntTreeNode root, int min, int max) {
      if (root != null) {
          if (root.data < min) {
              root = trim(root.right, min, max);
          } else if (root.data > max) {
              root = trim(root.left, min, max);
          } else {
              root.left  = trim(root.left,  min, max);
              root.right = trim(root.right, min, max);
          }
      }
      return root;
  }
  ```

-
  ```
  public void trim(int min, int max) {
      if (overallRoot != null) {
          while (overallRoot != null &&
                 (overallRoot.data < min || overallRoot.data > max)) {
              if (overallRoot.data < min) {
                  overallRoot = overallRoot.right;
              } else {
                  overallRoot = overallRoot.left;
              }
          }
          trim(overallRoot, min, max);
      }
  }
  private void trim(IntTreeNode root, int min, int max) {
      if (root != null) {
          while (root.left != null && root.left.data < min) {
              root.left = root.left.right;
          }
          while (root.right != null && root.right.data > max) {
              root.right = root.right.left;
          }
          trim(root.left, min, max);
          trim(root.right, min, max);
      }
  }
  ```

-
  ```
  public void tighten() {
      overallRoot = tighten(overallRoot);
  }
  private IntTreeNode tighten(IntTreeNode root) {
      if (root != null) {
          root.left = tighten(root.left);
          root.right = tighten(root.right);
          if (root.left != null && root.right == null) {
              root = root.left;
          } else if (root.left == null && root.right != null) {
              root = root.right;
          }
      }
      return root;
  }
  ```

-
  ```
  public IntTree combineWith(IntTree other) {
      IntTree result = new IntTree();
      result.overallRoot = combine(this.overallRoot, other.overallRoot);
      return result;
  }
  private IntTreeNode combine(IntTreeNode root1, IntTreeNode root2) {
      if (root1 == null) {
          if (root2 == null) {
              return null;
          } else {
              return new IntTreeNode(2, combine(null, root2.left),
                                        combine(null, root2.right));
          }
      } else {
          if (root2 == null) {
              return new IntTreeNode(1, combine(root1.left, null),
                                        combine(root1.right, null));
          } else {
              return new IntTreeNode(3, combine(root1.left, root2.left),
                                        combine(root1.right, root2.right));
          }
      }
  }
  ```

-
  ```
  public List<Integer> inOrderList() {
      List<Integer> result = new ArrayList<Integer>();
      inOrderList(overallRoot, result);
      return result;
  }
  private void inOrderList(IntTreeNode root, List<Integer> result) {
      if (root != null) {
          inOrderList(root.left, result);
          result.add(root.data);
          inOrderList(root.right, result);
      }
  }
  ```

-
  ```
  public void evenLevels() {
      overallRoot = evenLevels(overallRoot, 1);
  }

  private IntTreeNode evenLevels(IntTreeNode node, int height) {
      if (node!= null) {

          node.left = evenLevels(node.left, height + 1);
          node.right = evenLevels(node.right, height + 1);

          if(node.right == null && node.left == null && height % 2 == 1){
              node = null;
          }
      }
      return node;
  }
  ```

-
  ```
  public void makePerfect() {
      int height = height();
      overallRoot = makePerfect(overallRoot, height);
  }

  private IntTreeNode makePerfect(IntTreeNode root, int height) {
      if (height <= 0) {
          return root;
      }
      if (root == null) {
          root = new IntTreeNode(0);
      }
      root.left = makePerfect(root.left, height - 1);
      root.right = makePerfect(root.right, height - 1);
      return root;
  }

  public int height() {
      return height(overallRoot);
  }

  private int height(IntTreeNode root) {
      if (root == null) {
          return 0;
      } else {
          return 1 + Math.max(height(root.left), height(root.right));
      }
  }
  ```

Chapter 18

1.
```
public void addAll(HashIntSet other) {
    for (Node front : other.elementData) {
        Node current = front;
        while (current != null) {
            add(current.data);
            current = current.next;
        }
    }
}
```

2.
```
public boolean containsAll(HashIntSet other) {
    for (Node front : other.elementData) {
        Node current = front;
        while (current != null) {
            if (!contains(current.data)) {
                return false;
            }
            current = current.next;
        }
    }
    return true;
}
```

3.
```
public boolean equals(Object o) {
    if (!(o instanceof HashIntSet)) {
        return false;
    }
    HashIntSet other = (HashIntSet) o;
    for (Node front : elementData) {
        Node current = front;
        while (current != null) {
            if (!other.contains(current.data)) {
                return false;
            }
            current = current.next;
        }
    }
    return size == other.size();
}
```

4.
```
public void removeAll(HashIntSet other) {
    for (Node front : other.elementData) {
        Node current = front;
        while (current != null) {
            if (contains(current.data)) {
                remove(current.data);
            }
            current = current.next;
        }
    }
}
```

5.
```
public void retainAll(HashIntSet other) {
    for (int i = 0; i < elementData.length; i++) {
        if (elementData[i] != null) {
            while (elementData[i] != null && !other.contains(elementData[i].data)) {
                elementData[i] = elementData[i].next;
                size--;
            }
            Node current = elementData[i];
            while (current != null && current.next != null) {
                if (!other.contains(current.next.data)) {
                    current.next = current.next.next;
                    size--;
                } else {
                    current = current.next;
                }
            }
        }
    }
}
```

6.
```
public int[] toArray() {
    int[] result = new int[size];
    int i = 0;
    for (Node front : elementData) {
        Node current = front;
        while (current != null) {
            result[i] = current.data;
            i++;
            current = current.next;
        }
    }
    return result;
}
```

7.
```
public String toString() {
    if (isEmpty()) {
        return "[]";
    } else {
        String result = "[";
        for (Node front : elementData) {
            Node current = front;
            while (current != null) {
                if (result.length() > 1) {
                    result += ", ";
                }
                result += current.data;
                current = current.next;
            }
        }
        result += "]";
        return result;
    }
}
```

8.
```
public static void descending(int[] a) {
    Queue<Integer> pq = new PriorityQueue<Integer>(100, Collections.reverseOrder());
    for (int n : a) {
        pq.add(n);
    }
    for (int i = 0; i < a.length; i++) {
        a[i] = pq.remove();
    }
}
```

9.
```
public static int kthSmallest(PriorityQueue<Integer> pq, int k) {
    if (k <= 0 || k > pq.size()) {
        throw new IllegalArgumentException();
    }

    Queue<Integer> backup = new LinkedList<Integer>();
    int size = pq.size();
    int kth = 0;

    for (int i = 0; i < size; i++) {
        int n = pq.remove();
        if (i == k - 1) {
            kth = n;
        }
        backup.add(n);
    }

    while (!backup.isEmpty()) {
        pq.add(backup.remove());   // restore queue
    }

    return kth;
}
```

10.
```
public static boolean isConsecutive(PriorityQueue<Integer> pq) {
    if (pq.size() <= 1) {
        return true;
    }

    Queue<Integer> backup = new LinkedList<Integer>();
    int prev = pq.remove();
    backup.add(prev);
    boolean consecutive = true;

    while (!pq.isEmpty()) {
        int next = pq.remove();
        if (prev + 1 != next) {
            consecutive = false;
        }
        backup.add(next);
        prev = next;
    }

    while (!backup.isEmpty()) {
        pq.add(backup.remove());   // restore queue
    }

    return consecutive;
}
```

11.
```
public static void removeDuplicates(PriorityQueue<Integer> pq) {
    if (pq.size() <= 1) {
        return;
    }

    Queue<Integer> unique = new LinkedList<Integer>();
    int prev = pq.remove();
    unique.add(prev);

    while (!pq.isEmpty()) {
        int next = pq.remove();
        if (prev != next) {
            unique.add(next);
            prev = next;
        }
    }

    while (!unique.isEmpty()) {
        pq.add(unique.remove());   // restore queue
    }
}
```

12.
```
public static void stutter(PriorityQueue<Integer> pq) {
    Queue<Integer> temp = new LinkedList<Integer>();
    while (!pq.isEmpty()) {
        temp.add(pq.remove());
    }

    while (!temp.isEmpty()) {
        int n = temp.remove();
        pq.add(n);
        pq.add(n);
    }
}
```

13.
```
public int[] toArray() {
    int[] result = new int[size];
    for (int i = 0; i < size; i++) {
        result[i] = elementData[i + 1];
    }
    return result;
}
```

14.
```
public String toString() {
    if (isEmpty()) {
        return "[]";
    } else {
        String result = "[" + elementData[1];
        for (int i = 2; i <= size; i++) {
            result += ", " + elementData[i];
        }
        result += "]";
        return result;
    }
}
```

15.
```
public void merge(HeapIntPriorityQueue other) {
    for (int i = 1; i <= other.size; i++) {
        add(other.elementData[i]);
    }
}
```

Graphical User Interfaces (online supplement)

1.
```
import java.awt.*;
import javax.swing.*;
public class LayoutProblem1 {
    public static void main(String[] args) {
        JPanel center = new JPanel(new GridLayout(2, 2));
        center.add(new JButton("Button4"));
        center.add(new JButton("Button6"));
        center.add(new JButton("Button5"));
        center.add(new JButton("Button7"));

        JPanel north = new JPanel(new GridLayout(1, 3));
        north.add(new JButton("Button1"));
        north.add(new JButton("Button2"));
        north.add(new JButton("Button3"));

        JPanel south = new JPanel();
        south.add(new JLabel("Type stuff:"));
        south.add(new JTextField(10));

        JFrame frame = new JFrame("Good thing I studied!");
        frame.setSize(285, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(north, BorderLayout.NORTH);
        frame.add(center, BorderLayout.CENTER);
        frame.add(south, BorderLayout.SOUTH);
        frame.setVisible(true);
    }
}
```

2.
```
import java.awt.*;
import javax.swing.*;
public class LayoutProblem2 {
    public static void main(String[] args) {
        JPanel northgrid = new JPanel(new GridLayout(1, 3));
        northgrid.add(new JButton("hi"));
        northgrid.add(new JButton("long name"));
        northgrid.add(new JButton("bye"));

        JPanel north = new JPanel(new BorderLayout());
        north.add(new JLabel("Buttons:"), BorderLayout.WEST);
        north.add(northgrid);

        JPanel centereast = new JPanel(new GridLayout(4, 1));
        centereast.add(new JCheckBox("Bold"));
        centereast.add(new JCheckBox("Italic"));
        centereast.add(new JCheckBox("Underline"));
        centereast.add(new JCheckBox("Strikeout"));

        JPanel cc1 = new JPanel(new GridLayout(2, 2));
        cc1.add(new JButton("3"));
        cc1.add(new JButton("4"));
        cc1.add(new JButton("5"));
        cc1.add(new JButton("6"));

        JPanel centercenter = new JPanel(new GridLayout(2, 2));
        centercenter.add(new JButton("1"));
        centercenter.add(new JButton("2"));
        centercenter.add(cc1);
        centercenter.add(new JButton("7"));

        JPanel center = new JPanel(new BorderLayout());
        center.add(new JButton("Cancel"), BorderLayout.SOUTH);
        center.add(centercenter);
        center.add(centereast, BorderLayout.WEST);

        JFrame frame = new JFrame("Layout question");
        frame.setSize(450, 250);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new BorderLayout(2, 2));
        frame.add(north, BorderLayout.NORTH);
        frame.add(center, BorderLayout.CENTER);
        frame.setVisible(true);
    }
}
```

3.
```
import java.awt.*;
import javax.swing.*;
public class LayoutProblem3 {
    public static void main(String[] args) {
        JPanel bottom = new JPanel();
        bottom.add(new JRadioButton("Movies"));
        bottom.add(new JRadioButton("Music"));
        bottom.add(new JRadioButton("Videos"));
        bottom.add(new JRadioButton("DVD"));
        bottom.add(new JRadioButton("Web Pages"));
        bottom.add(new JRadioButton("Games"));
        bottom.add(new JRadioButton("News"));
        bottom.add(new JRadioButton("Shopping"));
        JPanel top = new JPanel(new BorderLayout());
        JPanel center = new JPanel(new BorderLayout());
        JPanel left = new JPanel(new GridLayout(5, 1));
        left.add(new JButton("Now Playing"));
        left.add(new JButton("Media Guide"));
        left.add(new JButton("Library"));
        left.add(new JButton("Help & Info"));
        left.add(new JButton("Services"));
        center.add(left, BorderLayout.WEST);
        center.add(new JTextArea(), BorderLayout.CENTER);
        JPanel northeast = new JPanel(new GridLayout(2, 2));
        for (int ii = 0; ii < 4; ii++) {
            northeast.add(new JButton(String.valueOf(ii)));
        }
        JPanel east = new JPanel(new BorderLayout());
        east.add(northeast, BorderLayout.NORTH);
        east.add(new JButton("OK"));
        JPanel lower = new JPanel();
        top.add(center, BorderLayout.CENTER);
        top.add(east, BorderLayout.EAST);
        JFrame frame = new JFrame("Midterm on Thursday!");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(bottom, BorderLayout.SOUTH);
        frame.add(top, BorderLayout.CENTER);
        frame.setVisible(true);
    }
}
```

4.
```
import java.awt.*;
import javax.swing.*;
public class LayoutProblem4 {
    public static void main(String[] args) {
        JPanel north = new JPanel(new BorderLayout());
        north.add(new JButton("1"), BorderLayout.WEST);
        north.add(new JButton("2"), BorderLayout.NORTH);
        north.add(new JButton("3"), BorderLayout.SOUTH);
        north.add(new JTextField("text"), BorderLayout.CENTER);

        JPanel center = new JPanel(new GridLayout(2, 2));
        center.add(new JButton("4"));
        JPanel five = new JPanel(new BorderLayout());
        five.add(new JButton("5"), BorderLayout.SOUTH);
        center.add(five);
        JPanel six = new JPanel();
        six.add(new JButton("6"));
        six.add(new JButton("7"));
        center.add(six);
        center.add(new JButton("8"));

        JFrame frame = new JFrame();
        frame.setTitle("I Dig Layout");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.add(north, BorderLayout.NORTH);
        frame.add(center, BorderLayout.CENTER);
        frame.add(new JLabel("Pretty tricky!"), BorderLayout.SOUTH);
        frame.setVisible(true);
    }
}
```

5.
```
import java.awt.*;
import javax.swing.*;
public class LayoutProblem5 {
    public static void main(String[] args) {
        JPanel north = new JPanel(new BorderLayout());
        JPanel northwest = new JPanel(new GridLayout(2, 1));
        northwest.add(new JLabel("To:"));
        northwest.add(new JLabel("CC:"));
        north.add(northwest, BorderLayout.WEST);
        JPanel northeast = new JPanel(new GridLayout(2, 1));
        northeast.add(new JTextField());
        northeast.add(new JTextField());
        north.add(northeast, BorderLayout.CENTER);

        JTextArea textArea = new JTextArea();

        JPanel south = new JPanel(new FlowLayout());
        south.add(new JButton("Send"));

        JFrame frame = new JFrame();
        frame.setTitle("Compose Message");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);
        frame.add(north, BorderLayout.NORTH);
        frame.add(textArea, BorderLayout.CENTER);
        frame.add(south, BorderLayout.SOUTH);
        frame.setVisible(true);
    }
}
```

6.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SillyStringGUI {
    public static void main(String[] args) {
        SillyStringGUI gui = new SillyStringGUI();
    }

    private JFrame frame;
    private JTextField textField;
    private JButton uppercase;
    private JButton lowercase;

    public SillyStringGUI() {
        textField = new JTextField(
            "The text can be made to all upper case or lower case");
        uppercase = new JButton("Upper Case");
        lowercase = new JButton("Lower Case");

        uppercase.addActionListener(new UpperCaseListener());
        lowercase.addActionListener(new LowerCaseListener());

        JPanel north = new JPanel(new FlowLayout());
        north.add(uppercase);
        JPanel south = new JPanel(new FlowLayout());
        south.add(lowercase);

        frame = new JFrame();
        frame.setTitle("Silly String Game");
        frame.setSize(300, 150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.add(north, BorderLayout.NORTH);
        frame.add(textField, BorderLayout.CENTER);
        frame.add(south, BorderLayout.SOUTH);
        frame.setVisible(true);
    }

    public class UpperCaseListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            textField.setText(textField.getText().toUpperCase());
        }
    }

    public class LowerCaseListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            textField.setText(textField.getText().toLowerCase());
        }
    }
}
```

7.
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MegaCalcGUI {
    public static void main(String[] args) {
        MegaCalcGUI calc = new MegaCalcGUI();
    }

    private JTextField operand1;
    private JTextField operand2;
    private JButton plus;
    private JButton clear;
    private JLabel result;
    private JFrame frame;

    public MegaCalcGUI() {
        operand1 = new JTextField(4);
        operand2 = new JTextField(4);
        plus = new JButton("+");
        clear = new JButton("Clear");
        result = new JLabel("?");

        CalcListener listener = new CalcListener();
        plus.addActionListener(listener);
        clear.addActionListener(listener);

        JPanel north = new JPanel();
        north.add(operand1);
        north.add(plus);
        north.add(operand2);

        frame = new JFrame();
        frame.setTitle("MegaCalcGUI");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(north, BorderLayout.NORTH);
        frame.add(result);
        frame.add(clear, BorderLayout.SOUTH);
        frame.pack();
        frame.setVisible(true);
    }

    public class CalcListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            Object source = event.getSource();
            if (source == plus) {
                try {
                    int op1 = Integer.parseInt(operand1.getText());
                    int op2 = Integer.parseInt(operand2.getText());
                    result.setText(String.valueOf(op1 + op2));
                }
                catch (NumberFormatException nfe) {
                    JOptionPane.showMessageDialog(frame, "Invalid number!");
                }
            }
            else if (source == clear) {
                operand1.setText("");
                operand2.setText("");
                result.setText("?");
            }
        }
    }
}
```

8.
```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class EyePanel extends JPanel {
    private int pupilY = 87;
    public EyePanel() {
        addMouseMotionListener(new EyeMouseMotionAdapter());
    }
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        drawEye(g2, 50, 50);
        drawEye(g2, 150, 50);
    }
    private void drawEye(Graphics2D g2, int eyex, int eyey) {
        g2.setColor(Color.WHITE);
        g2.fillOval(eyex, eyey, 100, 100);
        g2.setColor(Color.BLACK);
        g2.drawOval(eyex, eyey, 100, 100);
        g2.fillOval(eyex + 37, pupilY, 25, 25);
    }
    private class EyeMouseMotionAdapter extends MouseMotionAdapter {
        public void mouseMoved(MouseEvent event) {
            int my = event.getY();
            if (my < 50) {
                pupilY = 50;
            } else if (my > 150) {
                pupilY = 125;
            } else {
                pupilY = 87;
            }
            repaint();
        }
    }
}
```

```java
import java.awt.*;
import javax.swing.*;
public class EyeGUI {
    public static void main(String[] args) {
        EyeGUI gui = new EyeGUI();
    }

    private JFrame frame;
    private EyePanel panel;

    public EyeGUI() {
        panel = new EyePanel();

        frame = new JFrame();
        frame.setTitle("The eyes have it");
        frame.setSize(300, 250);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```

9.
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MovingLinePanel extends JPanel {
    private Point p1 = new Point(0, 0);
    private Point p2 = new Point(300, 300);
    private int dx = 5;
    public MovingLinePanel() {
        setBackground(Color.WHITE);
        Timer time = new Timer(20, new LineListener());
        time.start();
    }
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.BLUE);
        g.drawLine(p1.x, p1.y, p2.x, p2.y);
    }
    public class LineListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            p1.x += dx;
            p2.x -= dx;
            if (p1.x == 0 || p2.x == 0) {
                dx = -dx;
            }
            repaint();
        }
    }
}
```

```
import java.awt.*;
import javax.swing.*;
public class MovingLineGUI {
    public static void main(String[] args) {
        MovingLineGUI gui = new MovingLineGUI();
    }

    private JFrame frame;
    private MovingLinePanel panel;

    public MovingLineGUI() {
        panel = new MovingLinePanel();

        frame = new JFrame();
        frame.setTitle("Moving line");
        frame.setSize(300, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);
        frame.add(panel);
        frame.setVisible(true);
    }
}
```