Building Java Programs

Chapter 3
Lecture 3-1: Parameters

reading: 3.1
Redundant recipes

• Recipe for baking **20** cookies:
  • Mix the following ingredients in a bowl:
    • 4 cups flour
    • 1 cup butter
    • 1 cup sugar
    • 2 eggs
    • 1 bag chocolate chips ...
  • Place on sheet and Bake for about **10** minutes.

• Recipe for baking **40** cookies:
  • Mix the following ingredients in a bowl:
    • 8 cups flour
    • 2 cups butter
    • 2 cups sugar
    • 4 eggs
    • 2 bags chocolate chips ...
  • Place on sheet and Bake for about **10** minutes.
Parameterized recipe

- Recipe for baking 20 cookies:
  - Mix the following ingredients in a bowl:
    - 4 cups flour
    - 1 cup sugar
    - 2 eggs
    - ...

- Recipe for baking \( N \) cookies:
  - Mix the following ingredients in a bowl:
    - \( \frac{N}{5} \) cups flour
    - \( \frac{N}{20} \) cups butter
    - \( \frac{N}{20} \) cups sugar
    - \( \frac{N}{10} \) eggs
    - \( \frac{N}{20} \) bags chocolate chips ...
  - Place on sheet and Bake for about 10 minutes.

- **parameter**: A value that distinguishes similar tasks.
Consider the task of printing the following lines/boxes:

```
*************
*******
***********************************
**********
*        *
**********
*****
*   *
*   *
*****
```

Redundant figures
A redundant solution

public class Stars1 {
    public static void main(String[] args) {
        lineOf13();
        lineOf7();
        lineOf35();
        box10x3();
        box5x4();
    }

    public static void lineOf13() {
        for (int i = 1; i <= 13; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf7() {
        for (int i = 1; i <= 7; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void lineOf35() {
        for (int i = 1; i <= 35; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    public static void box10x3() {
        System.out.println("\n-------------------\n| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |
| * * * * * * * * * |

    }

    public static void box5x4() {
        System.out.println("\n-------------------\n| * * * * * | * * * * | * * * * | * * * * |
| * * * * * | * * * * | * * * * | * * * * |
| * * * * * | * * * * | * * * * | * * * * |
| * * * * * | * * * * | * * * * | * * * * |
| * * * * * | * * * * | * * * * | * * * * |

    }

    ...

• This code is redundant.
• Would variables help? Would constants help?
• What is a better solution?
  • line - A method to draw a line of any number of stars.
  • box - A method to draw a box of any size.
Parameterization

- **parameter**: A value passed to a method by its caller.

- Instead of `lineOf7`, `lineOf13`, write `line` to draw any length.
  - When *declaring* the method, we will state that it requires a parameter for the number of stars.
  - When *calling* the method, we will specify how many stars to draw.

```
main
  7
  13
line

**********
************
```
Declaring a parameter

Stating that a method requires a parameter in order to run

```java
public static void name (type name) {
    statement(s);
}
```

- Example:
  ```java
  public static void sayPassword(int code) {
      System.out.println("The password is: " + code);
  }
  ```
  
  - When `sayPassword` is called, the caller must specify the integer code to print.
Passing parameters

Calling a method and specifying values for its parameters

name (expression) ;

Example:

```java
public static void main(String[] args) {
    sayPassword(42);
    sayPassword(12345);
}
```

Output:

The password is 42
The password is 12345
Parameters and loops

- A parameter can guide the number of repetitions of a loop.

```java
public static void main(String[] args) {
    chant(3);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```

Output:
- Just a salad...
- Just a salad...
- Just a salad...
How parameters are passed

• When the method is called:
  • The value is stored into the parameter variable.
  • The method's code executes using that value.

```java
public static void main(String[] args) {
    chant(3);
    chant(7);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```
Common errors

- If a method accepts a parameter, it is illegal to call it without passing any value for that parameter.
  
  ```java
  chant(); // ERROR: parameter value required
  ```

- The value passed to a method must be of the correct type.
  
  ```java
  chant(3.7); // ERROR: must be of type int
  ```

- Exercise: Change the Stars program to use a parameterized method for drawing lines of stars.
Stars solution

// Prints several lines of stars.
// Uses a parameterized method to remove redundancy.
public class Stars2 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }
}
Multiple parameters

- A method can accept multiple parameters. (separate by , )
  - When calling it, you must pass values for each parameter.

- Declaration:
  
  ```java
  public static void name (type name, ..., type name) {
  statement(s);
  }
  ```

- Call:
  
  ```java
  methodName (value, value, ..., value);
  ```
Multiple parameters example

```java
public static void main(String[] args) {
    printNumber(4, 9);
    printNumber(17, 6);
    printNumber(8, 0);
    printNumber(0, 8);
}

public static void printNumber(int number, int count) {
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

Output:

```
444444444
171717171717
000000000
```

• Modify the Stars program to draw boxes with parameters.
Stars solution

// Prints several lines and boxes made of stars.
// Third version with multiple parameterized methods.

public class Stars3 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        for (int i = 1; i <= count; i++) {
            System.out.print("*");
        }
        System.out.println();
    }

    ...
Stars solution, cont'd.

...
A "Parameter Mystery" problem

```java
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 5;
        int y = 9;
        int z = 2;

        mystery(z, y, x);
        mystery(y, x, z);
    }

    public static void mystery(int x, int z, int y) {
        System.out.println(z + " " + y + " " + x);
    }
}
```
Strings

- **string**: A sequence of text characters.

  ```java
  String name = "text";
  String name = expression;
  ```

- Examples:

  ```java
  String name = "Marla Singer";
  int x = 3;
  int y = 5;
  String point = "(" + x + ", " + y + ")";
  ```
Strings as parameters

public class StringParameters {
    public static void main(String[] args) {
        String teacher = "Helene";
        sayHello(teacher);
        sayHello("Marty");
    }
    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}

Output:
Welcome, Helene
Welcome, Marty

- Modify the Stars program to use string parameters. Use a method named repeat that prints a string many times.
Stars solution

// Prints several lines and boxes made of stars.
// Fourth version with String parameters.

public class Stars4 {
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        repeat("*", count);
        System.out.println();
    }

    // Prints several lines and boxes made of stars.
    public static void main(String[] args) {
        line(13);
        line(7);
        line(35);
        System.out.println();
        box(10, 3);
        box(5, 4);
        box(20, 7);
    }

    // Prints the given number of stars plus a line break.
    public static void line(int count) {
        repeat("*", count);
        System.out.println();
    }
// Prints a box of stars of the given size.
public static void box(int width, int height) {
    line(width);
    for (int line = 1; line <= height - 2; line++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }
    line(width);
}

// Prints the given String the given number of times.
public static void repeat(String s, int times) {
    for (int i = 1; i <= times; i++) {
        System.out.print(s);
    }
}
Building Java Programs

Graphics

reading: Supplement 3G
videos: Ch. 3G #1-2
Graphical objects

We will draw graphics in Java using 3 kinds of objects:

- **DrawingPanel**: A window on the screen.
  - Not part of Java; provided by the authors.
- **Graphics**: A "pen" to draw shapes/lines on a window.
- **Color**: Colors in which to draw shapes.
Objects (briefly)

- **object**: An entity that contains data and behavior.
  - **data**: Variables inside the object.
  - **behavior**: Methods inside the object.
    - You interact with the methods; the data is hidden in the object.

- Constructing (creating) an object:
  ```
  type objectName = new type(parameters);
  ```

- Calling an object's method:
  ```
  objectName.methodName (parameters);
  ```
"Canvas" objects that represents windows/drawing surfaces

- To create a window:
  ```java
drawingPanel name = new DrawingPanel(width, height);
```

  Example:
  ```java
drawingPanel panel = new DrawingPanel(300, 200);
```

- The window has nothing on it.
  - We can draw shapes and lines on it using another object of type Graphics.
"Pen" objects that can draw lines and shapes

- Access it by calling `getGraphics` on your `DrawingPanel`.
  ```java
  Graphics g = panel.getGraphics();
  ```

- Draw shapes by calling methods on the `Graphics` object.
  ```java
  g.fillRect(10, 30, 60, 35);
  g.fillOval(80, 40, 50, 70);
  ```
Java class libraries, import

- **Java class libraries**: Classes included with Java's JDK.
  - organized into groups named *packages*
  - To use a package, put an *import declaration* in your program.

- **Syntax:**
  ```java
  // put this at the very top of your program
  import packageName.*;
  ```

- Graphics is in a package named *java.awt*
  ```java
  import java.awt.*;
  ```

- In order to use Graphics, you must place the above line at the very top of your program, before the *public class* header.
Coordinate system

- Each \((x, y)\) position is a *pixel* ("picture element").

- \((0, 0)\) is at the window's top-left corner.
  - \(x\) increases rightward and the \(y\) increases *downward*.

- The rectangle from \((0, 0)\) to \((200, 100)\) looks like this:

\[
\begin{array}{c}
(0, 0) \\
| \\
| \\
| \\
(200, 100)
\end{array}
\]
## Graphics methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>g.drawLine(x1, y1, x2, y2);</td>
<td>line between points (x1, y1), (x2, y2)</td>
</tr>
<tr>
<td>g.drawOval(x, y, width, height);</td>
<td>outline largest oval that fits in a box of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td>g.drawRect(x, y, width, height);</td>
<td>outline of rectangle of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td>g.drawString(text, x, y);</td>
<td>text with bottom-left at (x, y)</td>
</tr>
<tr>
<td>g.fillOval(x, y, width, height);</td>
<td>fill largest oval that fits in a box of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td>g.fillRect(x, y, width, height);</td>
<td>fill rectangle of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td>g.setColor(Color);</td>
<td>set Graphics to paint any following shapes in the given color</td>
</tr>
</tbody>
</table>
Color

- Create one using **Red-Green-Blue (RGB)** values from 0-255
  
  ```java
  Color name = new Color(red, green, blue);
  ```

- Example:
  
  ```java
  Color brown = new Color(192, 128, 64);
  ```

- Or use a predefined **Color** class constant (more common)
  
  ```java
  Color.CONSTANT_NAME
  ```

  where **CONSTANT_NAME** is one of:

  - BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, or YELLOW
Using Colors

- **Pass a** Color to Graphics object's `setColor` method
  - Subsequent shapes will be drawn in the new color.
    ```java
    g.setColor(Color.BLACK);
g.fillRect(10, 30, 100, 50);
g.drawLine(20, 0, 10, 30);
g.setColor(Color.RED);
g.fillOval(60, 40, 40, 70);
    ```

- **Pass a color to** DrawingPanel's `setBackground` method
  - The overall window background color will change.
    ```java
    Color brown = new Color(192, 128, 64);
    panel.setBackground(brown);
    ```
Outlined shapes

- To draw a colored shape with an outline, first *fill* it, then *draw* the same shape in the outline color.

```java
import java.awt.*; // so I can use Graphics

public class OutlineExample {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(150, 70);
        Graphics g = panel.getGraphics();

        // inner red fill
        g.setColor(Color.RED);
        g.fillRect(20, 10, 100, 50);

        // black outline
        g.setColor(Color.BLACK);
        g.drawRect(20, 10, 100, 50);
    }
}
```
Drawing with loops

- The $x,y, w,h$ expression can use the loop counter variable:

```java
DrawingPanel panel = new DrawingPanel(400, 300);
panel.setBackground(Color.YELLOW);
Graphics g = panel.getGraphics();

g.setColor(Color.RED);
for (int i = 1; i <= 10; i++) {
    g.fillOval(100 + 20 * i, 5 + 20 * i, 50, 50);
}
```

- Nested loops are okay as well:

```java
DrawingPanel panel = new DrawingPanel(250, 250);
Graphics g = panel.getGraphics();
g.setColor(Color.BLUE);

for (int x = 1; x <= 4; x++) {
    for (int y = 1; y <= 9; y++) {
        g.drawString("Java", x * 40, y * 25);
    }
}
```
Loops that begin at 0

- Beginning at 0 and using < can make coordinates easier.

Example:
- Draw ten stacked rectangles starting at (20, 20), height 10, width starting at 100 and decreasing by 10 each time:

```java
drawingPanel panel = new DrawingPanel(160, 160);
Graphics g = panel.getGraphics();

for (int i = 0; i < 10; i++) {
    g.drawRect(20, 20 + 10 * i, 100 - 10 * i, 10);
}
```
Drawing w/ loops questions

• Code from previous slide:

```java
DrawingPanel panel = new DrawingPanel(160, 160);
Graphics g = panel.getGraphics();

for (int i = 0; i < 10; i++) {
    g.drawRect(20, 20 + 10 * i, 100 - 10 * i, 10);
}
```

• Write variations of the above program that draw the figures at right as output.
Drawing w/ loops answers

• Solution #1:
  Graphics g = panel.getGraphics();
  for (int i = 0; i < 10; i++) {
    g.drawRect(20 + 10 * i, 20 + 10 * i, 
               100 - 10 * i, 10);
  }

• Solution #2:
  Graphics g = panel.getGraphics();
  for (int i = 0; i < 10; i++) {
    g.drawRect(110 - 10 * i, 20 + 10 * i, 
               10 + 10 * i, 10);
  }

Copyright 2008 by Pearson Education
Superimposing shapes

- When $\geq 2$ shapes occupy the same pixels, the last drawn "wins."

```java
import java.awt.*;
public class Car {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();

        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);

        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);

        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```
Drawing with methods

- To draw in multiple methods, you must pass `Graphics g`.

```java
import java.awt.*;

public class Car2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g);
    }

    public static void drawCar(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);

        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);

        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```
Parameterized figures

- Modify the car-drawing method so that it can draw cars at different positions, as in the following image.
  - Top-left corners: (10, 30), (150, 10)
import java.awt.*;

public class Car3 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(260, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30);
        drawCar(g, 150, 10);
    }

    public static void drawCar(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, 100, 50);
        g.setColor(Color.RED);
        g.fillOval(x + 10, y + 40, 20, 20);
        g.fillOval(x + 70, y + 40, 20, 20);
        g.setColor(Color.CYAN);
        g.fillRect(x + 70, y + 10, 30, 20);
    }
}
Drawing parameter question

- Modify `drawCar` to allow the car to be drawn at any size.
  - Existing car: size 100
  - Second car: size 50, top/left at (150, 10)

- Then use a `for` loop to draw a line of cars.
  - Start at (10, 130), each car size 40, separated by 50px.
import java.awt.*;
public class Car4 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(210, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30, 100);
        drawCar(g, 150, 10, 50);
        for (int i = 0; i < 5; i++) {
            drawCar(g, 10 + i * 50, 130, 40);
        }
    }
    public static void drawCar(Graphics g, int x, int y, int size) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, size, size / 2);
        g.setColor(Color.RED);
        g.fillOval(x + size / 10, y + 2 * size / 5,
                    size / 5, size / 5);
        g.fillOval(x + 7 * size / 10, y + 2 * size / 5,
                    size / 5, size / 5);
        g.setColor(Color.CYAN);
        g.fillRect(x + 7 * size / 10, y + size / 10,
                    3 * size / 10, size / 5);
    }
}
Polygon

Objects that represent arbitrary shapes

- Add points to a `Polygon` using its `addPoint(x, y)` method.

Example:

```java
DrawingPanel p = new DrawingPanel(100, 100);
Graphics g = p.getGraphics();
g.setColor(Color.GREEN);

Polygon poly = new Polygon();
poly.addPoint(10, 90);
poly.addPoint(50, 10);
poly.addPoint(90, 90);
g.fillPolygon(poly);
```
Animation with `sleep`

- **DrawingPanel's `sleep` method** pauses your program for a given number of milliseconds.

- **You can use `sleep` to create simple animations.**
  
  ```java
  DrawingPanel panel = new DrawingPanel(250, 200);
  Graphics g = panel.getGraphics();
  
  g.setColor(Color.BLUE);
  for (int i = 1; i <= 10; i++) {
    g.fillOval(15 * i, 15 * i, 30, 30);
    panel.sleep(500);
  }
  ```

- **Try adding `sleep` commands to loops in past exercises in this chapter and watch the panel draw itself piece by piece.**
Building Java Programs

Chapter 3
Lecture 3-2: Return; `double`; `System.out.printf`

reading: 3.2, 3.5, 4.4
videos: Ch. 3 #2, 4
Projectile problem

- Write a program that displays (as text and graphics) the paths of projectiles thrown at various velocities and angles.
  - Projectile #1: velocity = 60, angle = 50°, steps = 10
  - Projectile #2: velocity = 50, angle = 80°, steps = 50

<table>
<thead>
<tr>
<th>step</th>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>36.14</td>
<td>38.76</td>
<td>0.94</td>
</tr>
<tr>
<td>2</td>
<td>72.28</td>
<td>68.91</td>
<td>1.87</td>
</tr>
<tr>
<td>3</td>
<td>108.42</td>
<td>90.45</td>
<td>2.81</td>
</tr>
<tr>
<td>4</td>
<td>144.56</td>
<td>103.37</td>
<td>3.75</td>
</tr>
<tr>
<td>5</td>
<td>180.70</td>
<td>107.67</td>
<td>4.69</td>
</tr>
<tr>
<td>6</td>
<td>216.84</td>
<td>103.37</td>
<td>5.62</td>
</tr>
<tr>
<td>7</td>
<td>252.98</td>
<td>90.45</td>
<td>6.56</td>
</tr>
<tr>
<td>8</td>
<td>289.12</td>
<td>68.91</td>
<td>7.50</td>
</tr>
<tr>
<td>9</td>
<td>325.26</td>
<td>38.76</td>
<td>8.43</td>
</tr>
<tr>
<td>10</td>
<td>361.40</td>
<td>0.00</td>
<td>9.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>step</th>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1.74</td>
<td>9.69</td>
<td>0.20</td>
</tr>
<tr>
<td>2</td>
<td>3.49</td>
<td>18.98</td>
<td>0.40</td>
</tr>
</tbody>
</table>

...
Time observations

- We are given the number of "steps" of time to display.
  - We must figure out how long it takes the projectile to hit the ground, then divide this time into the # of steps requested.

<table>
<thead>
<tr>
<th>step</th>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>36.14</td>
<td>38.76</td>
<td>0.94</td>
</tr>
<tr>
<td>2</td>
<td>72.28</td>
<td>68.91</td>
<td>1.87</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>361.40</td>
<td>0.00</td>
<td>9.37</td>
</tr>
</tbody>
</table>

- Total time is based on the force of gravity on the projectile.
  - Force of gravity \((g) \approx 9.81 \text{ m/s}^2\), downward
  - The projectile has an initial upward velocity, which is fought by gravity until the projectile reaches its peak, then it falls.
Velocity and acceleration

- The projectile has a given initial velocity $v_0$, which can be divided into $x$ and $y$ components.
  - $v_{0x} = v_0 \cos \Theta$
  - $v_{0y} = v_0 \sin \Theta$
    - Example: If $v_0=13$ and $\Theta=60^\circ$, $v_{0x}=12$ and $v_{0y}=5$.

- The velocity $v_t$ of a moving body at time $t$, given initial velocity $v_0$ and acceleration $a$, can be expressed as:
  - $v_t = v_0 + a t$

- In our case, because of symmetry, at the end time $t$ the projectile is falling exactly as fast as it was first going up.
  - $v_t = -v_0$
    - $-v_0 = v_0 + a t$
    - $t = -2 \frac{v_0}{a}$
Return Values

reading: 3.2
self-check: #7-11
exercises: #4-6
videos: Ch. 3 #2
# Java's Math class

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>Math.round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>Math.ceil(value)</td>
<td>rounds up</td>
</tr>
<tr>
<td>Math.floor(value)</td>
<td>rounds down</td>
</tr>
<tr>
<td>Math.log10(value)</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>Math.max(value1, value2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>Math.min(value1, value2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>Math.pow(base, exp)</td>
<td>base to the exp power</td>
</tr>
<tr>
<td>Math.sqrt(value)</td>
<td>square root</td>
</tr>
<tr>
<td>Math.sin(value)</td>
<td>sine/cosine/tangent of an angle in radians</td>
</tr>
<tr>
<td>Math.cos(value)</td>
<td></td>
</tr>
<tr>
<td>Math.tan(value)</td>
<td></td>
</tr>
<tr>
<td>Math.toDegrees(value)</td>
<td>convert degrees to radians</td>
</tr>
<tr>
<td>Math.toRadians(value)</td>
<td>radians and back</td>
</tr>
<tr>
<td>Math.random()</td>
<td>random double between 0 and 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>PI</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
Calling Math methods

Math. methodName (parameters)

• Examples:

double squareRoot = Math.sqrt(121.0);
System.out.println(squareRoot); // 11.0

int absoluteValue = Math.abs(-50);
System.out.println(absoluteValue); // 50

System.out.println(Math.min(3, 7) + 2); // 5

• The Math methods do not print to the console.
  • Each method produces ("returns") a numeric result.
  • The results are used as expressions (printed, stored, etc.).
Return

- **return**: To send out a value as the result of a method.
  - The opposite of a parameter:
    - Parameters send information *in* from the caller to the method.
    - Return values send information *out* from a method to its caller.

```java
Math.abs(-42)  // -42
Math.round(2.71)  // 3
```
Math questions

• Evaluate the following expressions:
  • Math.abs(-1.23)
  • Math.pow(3, 2)
  • Math.pow(10, -2)
  • Math.sqrt(121.0) - Math.sqrt(256.0)
  • Math.round(Math.PI) + Math.round(Math.E)
  • Math.ceil(6.022) + Math.floor(15.9994)
  • Math.abs(Math.min(-3, -5))

Math.max and Math.min can be used to bound numbers. Consider an int variable named age.
  • What statement would replace negative ages with 0?
  • What statement would cap the maximum age to 40?
Returning a value

```java
public static type name(parameters) {
    statements;
    ...
    return expression;
}
```

- Example:

```java
// Returns the slope of the line between the given points.
public static double slope(int x1, int y1, int x2, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    return dy / dx;
}
```
Return examples

// Converts Fahrenheit to Celsius.
public static double fToC(double degreesF) {
    double degreesC = 5.0 / 9.0 * (degreesF - 32);
    return degreesC;
}

// Computes triangle hypotenuse length given its side lengths.
public static double hypotenuse(int a, int b) {
    double c = Math.sqrt(a * a + b * b);
    return c;
}

• You can shorten the examples by returning an expression:

    public static double fToC(double degreesF) {
        return 5.0 / 9.0 * (degreesF - 32);
    }
Common error: Not storing

- Many students incorrectly think that a `return` statement sends a variable's name back to the calling method.

```java
class SlopeCalculator {
    public static double slope(int x1, int x2, int y1, int y2) {
        double dy = y2 - y1;
        double dx = x2 - x1;
        double result = dy / dx;
        return result;
    }
}
```

```java
public static void main(String[] args) {
    slope(0, 0, 6, 3);
    System.out.println("The slope is "+ result);  // ERROR: result not defined
}
```
Fixing the common error

• Instead, returning sends the variable's *value* back.
  • The returned value must be stored into a variable or used in an expression to be useful to the caller.

```java
public static void main(String[] args) {
    double s = slope(0, 0, 6, 3);
    System.out.println("The slope is " + s);
}

public static double slope(int x1, int x2, int y1, int y2) {
    double dy = y2 - y1;
    double dx = x2 - x1;
    double result = dy / dx;
    return result;
}
```
Quirks of real numbers

- Some `Math` methods return `double` or other non-`int` types.
  ```java
  int x = Math.pow(10, 3);  // ERROR: incompat. types
  ```

- Some `double` values print poorly (too many digits).
  ```java
  double result = 1.0 / 3.0;
  System.out.println(result);  // 0.333333333333333333333333333333
  ```

- The computer represents `doubles` in an imprecise way.
  ```java
  System.out.println(0.1 + 0.2);
  ```
  - Instead of 0.3, the output is `0.30000000000000004`
Type casting

- **type cast**: A conversion from one type to another.
  - To promote an `int` into a `double` to get exact division from `/`
  - To truncate a `double` from a real number to an integer

**Syntax:**

```
(type) expression
```

**Examples:**
```
double result = (double) 19 / 5;     // 3.8
int result2 = (int) result;          // 3
int x = (int) Math.pow(10, 3);       // 1000
```
More about type casting

- Type casting has high precedence and only casts the item immediately next to it.
  - `double x = (double) 1 + 1 / 2; // 1`
  - `double y = 1 + (double) 1 / 2; // 1.5`

- You can use parentheses to force evaluation order.
  - `double average = (double) (a + b + c) / 3;`

- A conversion to `double` can be achieved in other ways.
  - `double average = 1.0 * (a + b + c) / 3;`
System.out.printf

an advanced command for printing formatted text

System.out.printf("format string", parameters);

- A format string contains **placeholders** to insert parameters into it:
  - `%d` an integer
  - `%f` a real number
  - `%s` a string

- Example:
  ```java
  int x = 3;
  int y = 2;
  System.out.printf("(%d, %d)\n", x, y);   // (3, 2)
  ```
System.out.printf cont'd

• A placeholder can specify the parameter's width or precision:
  • %8d an integer, 8 characters wide, right-aligned
  • %-8d an integer, 8 characters wide, left-aligned
  • %.4f a real number, 4 characters after decimal
  • %6.2f a real number, 6 characters wide, 2 after decimal

• Examples:
  int age = 45;
  double gpa = 1.2345678;

  System.out.printf("%8d %4f\n", age, gpa);
  System.out.printf("%8.3f %.1f %.5f", gpa, gpa, gpa);

• Output:
  45      1.23
  1.234 1.2 1.23457
Projectile problem revisited

*Recall:* Display (as text and graphics) the paths of projectiles thrown at various velocities and angles.

- Projectile #1: velocity = 60, angle = 50°, steps = 10
- Projectile #2: velocity = 50, angle = 80°, steps = 50

<table>
<thead>
<tr>
<th>step</th>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>36.14</td>
<td>38.76</td>
<td>0.94</td>
</tr>
<tr>
<td>2</td>
<td>72.28</td>
<td>68.91</td>
<td>1.87</td>
</tr>
<tr>
<td>3</td>
<td>108.42</td>
<td>90.45</td>
<td>2.81</td>
</tr>
<tr>
<td>4</td>
<td>144.56</td>
<td>103.37</td>
<td>3.75</td>
</tr>
<tr>
<td>5</td>
<td>180.70</td>
<td>107.67</td>
<td>4.69</td>
</tr>
<tr>
<td>6</td>
<td>216.84</td>
<td>103.37</td>
<td>5.62</td>
</tr>
<tr>
<td>7</td>
<td>252.98</td>
<td>90.45</td>
<td>6.56</td>
</tr>
<tr>
<td>8</td>
<td>289.12</td>
<td>68.91</td>
<td>7.50</td>
</tr>
<tr>
<td>9</td>
<td>325.26</td>
<td>38.76</td>
<td>8.43</td>
</tr>
<tr>
<td>10</td>
<td>361.40</td>
<td>0.00</td>
<td>9.37</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>step</th>
<th>x</th>
<th>y</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>1.74</td>
<td>9.69</td>
<td>0.20</td>
</tr>
<tr>
<td>2</td>
<td>3.49</td>
<td>18.98</td>
<td>0.40</td>
</tr>
</tbody>
</table>

...
Based on the previous, we can now display $x$ and $time$.

- $x_t = v_x \cdot t$ since there is no force in the $x$ direction.

<table>
<thead>
<tr>
<th>step</th>
<th>$x$</th>
<th>$y$</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>????</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>36.14</td>
<td>????</td>
<td>0.94</td>
</tr>
<tr>
<td>2</td>
<td>72.28</td>
<td>????</td>
<td>1.87</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>361.40</td>
<td>????</td>
<td>9.37</td>
</tr>
</tbody>
</table>

To display the $y$, we need to compute the projectile's displacement in $y$ direction at each time increment.

- $y_t = v_{0y} \cdot t + \frac{1}{2} a \cdot t^2$

Since this formula is complicated, let's make it into a method.
// This program computes and draws the trajectory of a projectile.

import java.awt.*;

public class Projectile {

    // constant for Earth's gravity acceleration in meters/second^2
    public static final double ACCELERATION = -9.81;

    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(420, 250);
        Graphics g = panel.getGraphics();

        // v0  angle  steps
        table(g,  60,  50,  10);
        g.setColor(Color.RED);
        table(g,  50,  80,  50);
    }

    // returns the displacement for a body under acceleration
    public static double displacement(double v0, double t, double a) {
        return v0 * t + 0.5 * a * t * t;
    }

    ...
}
// prints a table showing the trajectory of an object given its initial velocity v and angle and number of steps
public static void table(Graphics g, double v0, double angle, int steps) {
    double v0x = v0 * Math.cos(Math.toRadians(angle));
    double v0y = v0 * Math.sin(Math.toRadians(angle));
    double totalTime = -2.0 * v0y / ACCELERATION;
    double dt = totalTime / steps;
    System.out.println("    step       x       y    time");
    for (int i = 0; i <= steps; i++) {
        double time = i * dt;
        double x = i * v0x * dt;
        double y = displacement(v0y, time, ACCELERATION);
        System.out.printf("%8d%8.2f%8.2f%8.2f\n", i, x, y, time);
        g.fillOval((int) x, (int) (250 - y), 5, 5);
    }
}
Interactive programs

• We have written programs that print console output, but it is also possible to read *input* from the console.
  • The user types input into the console. We capture the input and use it in our program.
  • Such a program is called an *interactive program*.

• Interactive programs can be challenging.
  • Computers and users think in very different ways.
  • Users misbehave.
Input and `System.in`

- `System.out`
  - An object with methods named `println` and `print`

- `System.in`
  - not intended to be used directly
  - We use a second object, from a class `Scanner`, to help us.

- **Constructing a `Scanner` object to read console input:**
  
  ```java
  Scanner name = new Scanner(System.in);
  ```

  - Example:
    ```java
    Scanner console = new Scanner(System.in);
    ```
Java class libraries, import

- **Java class libraries**: Classes included with Java's JDK.
  - organized into groups named *packages*
  - To use a package, put an *import declaration* in your program.

- **Syntax**:
  
  ```java
  // put this at the very top of your program
  import packageName.*;
  ```

- **Scanner is in a package named java.util**
  
  ```java
  import java.util.*;
  ```

- **To use Scanner, you must place the above line at the top of your program (before the public class header).**
**Scanner methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextInt()</code></td>
<td>reads a token of user input as an <code>int</code></td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>reads a token of user input as a <code>double</code></td>
</tr>
<tr>
<td><code>next()</code></td>
<td>reads a token of user input as a <code>String</code></td>
</tr>
<tr>
<td><code>nextLine()</code></td>
<td>reads a <em>line</em> of user input as a <code>String</code></td>
</tr>
</tbody>
</table>

- Each method waits until the user presses Enter.
  - The value typed is returned.

```java
System.out.print("How old are you? ");  // prompt
int age = console.nextInt();
System.out.println("You'll be 40 in " +
   (40 - age) + " years.");
```

- **prompt**: A message telling the user what input to type.
Example Scanner usage

```java
import java.util.*; // so that I can use Scanner

public class ReadSomeInput {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("How old are you? ");
        int age = console.nextInt();
        System.out.println(age + "... That's quite old!");
    }
}
```

- Output (user input underlined):
  
  How old are you? 14
  14... That's quite old!
Another Scanner example

import java.util.*;  // so that I can use Scanner

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

        System.out.print("Please type three numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();
        int num3 = console.nextInt();

        int sum = num1 + num2 + num3;
        System.out.println("The sum is "+ sum);
    }
}

- Output (user input underlined):
  Please type three numbers: 8 6 13
  The sum is 27

- The Scanner can read multiple values from one line.
Input tokens

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by *whitespace* (spaces, tabs, newlines).
  - How many tokens appear on the following line of input?
    23  John Smith  42.0  "Hello world"  $2.50  " 19"

- When a token is not the type you ask for, it crashes.

```
System.out.print("What is your age? ");
int age = console.nextInt();
```

**Output:**

```
What is your age? Timmy
java.util.InputMismatchException
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    ...
```

Scanners as parameters

- If many methods read input, declare a Scanner in main and pass it to the others as a parameter.

```java
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    System.out.println("The sum is " + sum);
}

// Prompts for 3 numbers and returns their sum.
public static int readSum3(Scanner console) {
    System.out.print("Type 3 numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```
Cumulative sum

**reading:** 4.1
**self-check:** Ch. 4 #1-3
**exercises:** Ch. 4 #1-6
Adding many numbers

• How would you find the sum of all integers from 1-1000?
  
  int sum = 1 + 2 + 3 + 4 + ... ;
  System.out.println("The sum is " + sum);

• What if we want the sum from 1 - 1,000,000?
  Or the sum up to any maximum?

• We could write a method that accepts the max value as a parameter and prints the sum.
  • How can we generalize code like the above?
A failed attempt

• An incorrect solution for summing 1-1000:

```java
for (int i = 1; i <= 1000; i++) {
    int sum = 0;
    sum = sum + i;
}

// sum is undefined here
System.out.println("The sum is " + sum);
```

• **sum's scope is in the for loop**, so the code does not compile.

• **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
  
  • The **sum** in the above code is an attempt at a cumulative sum.
Fixed cumulative sum loop

• A corrected version of the sum loop code:

```java
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

Key idea:

• Cumulative sum variables must be declared outside the loops that update them, so that they will exist after the loop.
Cumulative product

• This cumulative idea can be used with other operators:

```java
int product = 1;
for (int i = 1; i <= 20; i++) {
    product = product * 2;
}
System.out.println("2 ^ 20 = " + product);
```

• How would we make the base and exponent adjustable?
Scanner and cumulative sum

- We can do a cumulative sum of user input:

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

Scanner console = new Scanner(System.in);
System.out.print("How many numbers to add? ");
int count = console.nextInt();

int sum = 0;
for (int i = 1; i <= count; i++) {
    System.out.print("Type a number: ");
    sum = sum + console.nextInt();
}
System.out.println("The sum is " + sum);

• Output:

How many numbers to add? 3
Type a number: 2
Type a number: 6
Type a number: 3
The sum is 11
Cumulative sum question

- Write a program that reads two employees' hours and displays each employee's total and the overall total hours.
  - The company doesn't pay overtime; cap each day at 8 hours.

- Example log of execution:

  Employee 1: How many days? 3
  Hours? 6
  Hours? 12
  Hours? 5
  Employee 1's total hours = 19 (6.3 / day)

  Employee 2: How many days? 2
  Hours? 11
  Hours? 6
  Employee 2's total hours = 14 (7.0 / day)

  Total hours for both = 33
import java.util.*;

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

        int hours1 = processEmployee(console, 1);
        int hours2 = processEmployee(console, 2);

        int total = hours1 + hours2;
        System.out.println("Total hours for both = " + total);
    }

    // Computes the total paid hours worked by two employees.
    // The company does not pay for more than 8 hours per day.
    // Uses a "cumulative sum" loop to compute the total hours.
    static int processEmployee(Scanner console, int employee) {
        int hours = 0;
        while (true) {
            System.out.print("Enter hours for employee "+employee+": ");
            int input = console.nextInt();
            if (input > 8) {
                System.out.println("More than 8 hours!");
            } else {
                hours += input;
                break;
            }
        }
        return hours;
    }
}

// Reads hours information about an employee with the given number.
// Returns total hours worked by the employee.
public static int processEmployee(Scanner console, int number) {
    System.out.print("Employee "+ number + ": How many days? ");
    int days = console.nextInt();

    // totalHours is a cumulative sum of all days' hours worked.
    int totalHours = 0;
    for (int i = 1; i <= days; i++) {
        System.out.print("Hours? ");
        int hours = console.nextInt();
        totalHours = totalHours + Math.min(hours, 8);
    }

    double hoursPerDay = (double) totalHours / days;
    System.out.printf("Employee %d's total hours = %d (%.1f / day)\n", number, totalHours, hoursPerDay);
    System.out.println();
    return totalHours;
}
Cumulative sum question

- Write a modified version of the Receipt program from Ch.2 that prompts the user for how many people ate and how much each person's dinner cost.
  - Display results in format below, with $ and 2 digits after the .

Example log of execution:

How many people ate? 4
Person #1: How much did your dinner cost? 20.00
Person #2: How much did your dinner cost? 15
Person #3: How much did your dinner cost? 25.0
Person #4: How much did your dinner cost? 10.00

Subtotal: $70.00
Tax: $5.60
Tip: $10.50
Total: $86.10
// This program enhances our Receipt program using a cumulative sum.
import java.util.*;

public class Receipt2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("How many people ate? ");
        int people = console.nextInt();
        double subtotal = 0.0; // cumulative sum
        for (int i = 1; i <= people; i++) {
            System.out.print("Person #" + i + ": How much did your dinner cost? ");
            double personCost = console.nextDouble();
            subtotal = subtotal + personCost; // add to sum
        }
        results(subtotal);
    }

    // Calculates total owed, assuming 8% tax and 15% tip
    public static void results(double subtotal) {
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;
        System.out.printf("Subtotal: $%.2f\n", subtotal);
        System.out.printf("Tax: $%.2f\n", tax);
        System.out.printf("Tip: $%.2f\n", tip);
        System.out.printf("Total: $%.2f\n", total);
    }
}
The **if** statement

*Executes a block of statements only if a test is true*

```java
if (test) {
    statement;
    ...
    statement;
}
```

- **Example:**
  
  ```java
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Application accepted.");
}
```
The **if/else** statement

Executes one block if a test is true, another if false

```java
if (test) {
    statement(s);
} else {
    statement(s);
}
```

**Example:**

```java
double gpa = console.nextDouble();
if (gpa >= 2.0) {
    System.out.println("Welcome to Mars University!");
} else {
    System.out.println("Application denied.");
}
```
Relational expressions

- A **test** in an `if` is the same as in a `for` loop.

```java
for (int i = 1; i <= 10; i++) { ... 
if (i <= 10) { ... 
```

- These are **boolean** expressions, seen in Ch. 5.

- Tests use **relational operators**:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>3.2 != 2.5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>126 &lt;= 100</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>5.0 &gt;= 5.0</td>
<td>true</td>
</tr>
</tbody>
</table>
Logical operators: &&, ||, !

- Conditions can be combined using logical operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>(2 == 3) &amp;&amp; (-1 &lt; 5)</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>!</td>
<td>not</td>
<td>!(2 == 3)</td>
<td>true</td>
</tr>
</tbody>
</table>

- "Truth tables" for each, used with logical values p and q:

| p   | q   | p && q | p || q |
|-----|-----|--------|--------|
| true| true| true   | true   |
| true| false| false  | true   |
| false| true| false  | true   |
| false| false| false  | false  |

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>
Evaluating logic expressions

- Relational operators have lower precedence than math.
  \[
  5 \times 7 \geq 3 + 5 \times (7 - 1) \\
  5 \times 7 \geq 3 + 5 \times 6 \\
  35 \geq 3 + 30 \\
  35 \geq 33 \\
  \text{true}
  \]

- Relational operators cannot be "chained" as in algebra.
  \[
  2 \leq x \leq 10 \\
  \text{true} \leq 10 \\
  \text{error!}
  \]

- Instead, combine multiple tests with && or ||
  \[
  2 \leq x \land x \leq 10 \\
  \text{true} \land \text{false} \\
  \text{false}
  \]
Logical questions

- What is the result of each of the following expressions?

```java
int x = 42;
int y = 17;
int z = 25;

- y < x && y <= z
- x % 2 == y % 2 || x % 2 == z % 2
- x <= y + z && x >= y + z
- !(x < y && x < z)
- (x + y) % 2 == 0 || !(z - y) % 2 == 0
```

- Answers: true, false, true, true, false