Recap

- Nested loops
  - If a loop is inside another loop, we say the two form a nested loop
  - The number of loop times of the inner loop can depend on the outer loop variable
- Method with parameters
  - Specify parameters to control the behavior of a method
  - Redundancy removal/abstraction through generalization

Example: Nested Loop

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

Example: Rewrite

```java
public static void dots(int n) {
    for (int i = 1; i <= n; i++) {
        System.out.print(".");
    }
}

public static void main(String[] args) {
    int N = 5;
    for (int line = 1; line <= N; line++) {
        dots(-1 * line + N);
        System.out.print(line);
        dots(line - 1);
        System.out.println();
    } // end of outer for loop
}
```

Out-Class Read: Another Way to Motivate Method With Parameters
Example: Reuse Matrix

Print two copies of inverse diagonal matrix

```
1.1
2.
3.
...1
...2.
...3.
4...
5....
```

Solution 1

This code is redundant.

```
public class DrawMatrix1 {
    public static void main(String[] args) {
        drawMatrix3();
        drawMatrix5();
    }
    public static void drawMatrix3() {
        int N = 3;
        for (int line = 1; line <= N; line++) {
            // initial dots
            for (int j = 1; j <= (-1 * line + N); j++) {
                System.out.print(".");
            }
            // the number
            System.out.print(line);
            // the second set of dots
            for (int j = 1; j <= (line - 1); j++) {
                System.out.print(".");
            }
            System.out.println();
        }
    }
    public static void drawMatrix5() {
        int N = 5;
        for (int line = 1; line <= N; line++) {
            // initial dots
            for (int j = 1; j <= (-1 * line + N); j++) {
                System.out.print(".");
            }
            System.out.print(line);
            // the number
            for (int j = 1; j <= (line - 1); j++) {
                System.out.print(".");
            }
            System.out.println();
        }
    }
}
```

Solution 2

```
public class DrawMatrix2 {
    public static void drawMatrix() {
        for (int line = 1; line <= N; line++) {
            for (int j = 1; j <= (-1 * line + N); j++) {
                // initial dots
                System.out.print(".");
            }
            System.out.print(line);
            // the number
            for (int j = 1; j <= (line - 1); j++) {
                // the second set of dots
                System.out.print(".");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        drawMatrix(); // should print 3
        drawMatrix(); // should print 5
    }
}
```

Solution 3: Parameterization

Specify a parameter to control the behavior of a method
- Methods with parameters solve an entire class of similar problems

Redundancy removal/abstraction through generalization
- The more general a building block, the easier to reuse it
- We will learn more techniques on generalization/abstraction
Method Design Heuristics

1. The main method should read as a concise summary of the overall set of tasks performed by the program.
2. Each method should have a clear set of responsibilities.
3. No method should do too large a share of the overall task.
4. Use method with parameters to remove redundancy with generalization, but do not over generalize.
5. Data should be declared/used at the lowest level possible (localized data).

Example: Draw Boxes

Consider the task of printing the following box figures using "*" and " ":

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

What method(s) and parameter(s)?

Box Figures Param. Chaining

```
public static void drawBox(int width, int height) {
    // step 1. print width of "*
    repeat("*", width);
    System.out.println();
    // step 2. loop a total of height - 2 times
    //     for each row:
    //       print *
    //       print width -2 " 
    for (int i = 1; i <= height - 2; i++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }
    // step 3. print width of "*
    repeat("*", width);
    System.out.println();
} // end of drawBox
```

Method Correctness Analysis/Testing

Q: How do you analyze/test the correctness of a method?

Basic idea

- A method is correct if for each input (parameters), the output is what you expect for the input.

An issue:

- Some input may not make sense.
- Two approaches
  - Detect such bad input and report
  - Assume that the input is in the valid range: called precondition of the method

Recap: Graphics using StdDraw

To access a static method or class variable defined in another class, use <classname>.<method-name>(...), for example,

```
StdDraw.setCanvasSize(100, 100);
Color.BLACK;
```

Method designers may design different methods for the same function

```
StdDraw: rectangle(double x, double y, double halfWidth, double halfHeight);
DrawingPanel (Java): rect (int x0, int y0, int width, int height);
```
Example: Parameterized Drawing

- Write method `drawCar`:
  - Center: \((x_0 + 100/2, y_0 + 50/2)\)
  - Size: 50, 25

- Center: \((x_0 + 15 + 10, y_0)\)
- Radius: 10

- Center: \((x_0 + 100 - 15 - 10, y_0)\)
- Radius: 10

- Center: \((x_0 + 100 - 30/2, y_0 + 50/2)\)
- Size: 15, 10

```java
public static void drawCar(int x0, int y0) {
    StdDraw.setPenColor(Color.BLACK);
    StdDraw.filledRectangle(x0 + 100/2, y0 + 50/2, 100/2, 50/2);
    // draw wheels
    StdDraw.setPenColor(Color.RED);
    StdDraw.filledCircle(x0 + 15 + 10, y0, 10);
    StdDraw.filledCircle(x0 + 100 - 15 - 10, y0, 10);
    // draw window
    StdDraw.setPenColor(Color.CYAN);
    StdDraw.filledRectangle(x0 + 100 - 30 / 2, y0 + 25, 15, 10);
}
```

Criticism of the method?

- See `Car.java` for the final version.

Example: Drawing using Loops

- You may use loop (nested loop) to produce more "complex" patterns:

```java
public static void main(String[] args) {
    init(); // set x, y scale to 512
    int x0 = 56, y0 = 56;
    int size = 50;
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            StdDraw.square(x0 + i * size + size / 2,
                           y0 + j * size + size / 2,
                           size/2);
        }
    }
}
```

What is the display?

Example: Book Cover

- White 500x600 drawing panel
- Three components at
  - (20, 415), (165, 330), (220, 85) with sizes 150, 120, and 240
- Each component
  - Yale blue background
  - white "CS112" text left @ 1/2 horizontal, 4/5 vertical
  - 10 brown (red=192, green=128, blue=64) "bricks"
    - 1 pixel between two adjacent bricks

```java
BookCover.java
```

Summary of Progress So Far

- Primitive data types, expressions and variables
- for loops
- Methods with parameters
**Motivating Example: Car Launch**

- Two cars launching from different heights, which lands further?
  - Car 1: initial height: 600m, horizontal speed: 30 m/sec, vertical speed: 20 m/sec
  - Car 2: initial height: 500m, horizontal speed: 40 m/sec, vertical speed: 30 m/sec

- Write a program to display their trajectories in 10 sec

---

**Background: Physics**

- In physics, for each dimension (x or y), given initial velocity \( v_0 \) in m/s, acceleration \( a \) in m/s\(^2\), and elapsed time \( t \) in s
  - the speed of the moving body at time \( t \):
    \[ V = v_0 + a \cdot t \]
  - The displacement of the moving body at time \( t \):
    \[ \text{Displacement} = v_0 \cdot t + \frac{1}{2} a \cdot t^2 \]
  - The position of the moving body at time \( t \):
    \[ \text{Pos} = \text{Initial position} + \text{displacement} \]

---

**Position**

- \[ P_0 + v_0 \cdot t + \frac{1}{2} a \cdot t^2 \]

- Horizontal (x):
  - \[ v_{x0} \cdot t \]

- Vertical (y):
  - \[ g \approx 9.81 \text{ m/s}^2, \text{downward} \]
  - \[ y_0 + v_{y0} \cdot t - \frac{1}{2} g \cdot t^2 \]

---

**Example**

- See CarLaunchv1.java

---

**Exercise: Generalization**

- Consider the task of printing the following lines/boxes:
  
  *************                          //13
  **********
  **********
  *******
  *******
  *******
  *******
  *******
  *******
  *******
  *******

---

**Backup Slides**
Design I

- Provide a new method `drawLine()` by calling `repeat()`

```java
public static void drawLine(int width) {
    repeat("*", width);
} // end of drawLine
```

Design II

- Consider a line as a special box with height

```java
public static void drawBox(int width, int height) {
    // super elastic drawBox
    // (int width, int height)
    if (height >= 2) // a real box
        draw rest
```

The if statement

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

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

- Example:
  ```java
  if (gpa >= 2.0) {
      System.out.println("Application accepted.");
  }
  ```

The if-else statement

- An else clause can be added to an if statement to make it an if-else statement:

```java
if (test) {
    statement1;
} else {
    statement2;
}
```

- If the condition is true, statement1 is executed; if the condition is false, statement2 is executed.
- One or the other will be executed, but not both.

- Example:
  ```java
  if (gpa >= 2.0) {
      System.out.println("Welcome to Mars University!");
  } else {
      System.out.println("Application denied.");
  }
  ```

Design II

- Consider a line as a special box

```java
public static void drawBox(int width, int height) {
    // super elastic drawBox
    public static void drawBox(int width, int height) {
        // step 1. print width of "*
        repeat("*", width);
        System.out.println();
        for (int i = 1; i <= height - 2; i++) {
            System.out.print("*");
            repeat(" ", width - 2);
            System.out.println("*");
        }
        if (height >= 2) // a real box
            draw rest
    }
    // end of drawBox
```
public static void drawLine(int width) {
    repeat("*", width);
} // end of drawLine

public static void drawBox(int width, int height) {
    repeat("*", width);
    System.out.println();
    for (int i = 1; i <= height - 2; i++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }
    if (height >= 2) {
        repeat("*", width);
        System.out.println();
    } // end of if
} // end of drawBox

public static void drawBox(int width, int height) {
    repeat("*", width);
    System.out.println();
    for (int i = 1; i <= height - 2; i++) {
        System.out.print("*");
        repeat(" ", width - 2);
        System.out.println("*");
    }
    if (height >= 2) {
        repeat("*", width);
        System.out.println();
    } // end of if
} // end of drawBox