`Scanner` objects

Readings: 3.4

Interactive programs

- We have written programs that print console output.
- It is also possible to read *input* from the console.
  - The user types the input into the console.
  - The program uses the input to do something.
  - Such a program is called an *interactive program*.

Interactive programs

- Interactive programs can be challenging.
  - Computers and users think in very different ways.
  - Users tend to "misbehave".

Input and `System.in`

- `System.out` is an object!
  - It has the methods named `println` and `print` for printing to the console.
- We read input using an object named `System.in`
  - `System.in` is not intended to be used directly.
  - We will use another object, from a class called `Scanner`, to read input from `System.in`.

`Scanner`

- Constructing a `Scanner` object to read the console:

```
Scanner <name> = new Scanner(System.in);
```

- Example:

```
Scanner console = new Scanner(System.in);
```


`Scanner` methods

- Some methods of `Scanner`:

Method	Description
`nextInt()`	reads and returns user input as an `int`
`nextDouble()`	reads and returns user input as a `double`
`next()`	reads and returns user input as a `String`

- Each of these methods pauses your program until the user types input and presses Enter.
  - Then the value typed is *returned* to your program.

Using a `Scanner` object

- Example:

```
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 printed to the user, telling them what input to type.

Input tokens

- **token**: A unit of user input, as read by the `Scanner`.
  - Tokens are separated by whitespace (spaces, tabs, new lines).
  - How many tokens appear on the following line of input?
    `23  John Smith  42.0 "Hello world"`
- When the token doesn't match the type the `Scanner` tries to read, the program crashes.
  - Example:

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

Sample Run:

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


Importing classes

- **Java class libraries**: A large set of Java classes available for you to use.
  - Classes are grouped into *packages*.
  - To use the classes from a package, you must include an *import declaration* at the top of your program.
- Import declaration, general syntax:

```
import <package name>.*;
```

- `Scanner` is in a package named `java.util`

```
import java.util.*;
```


A complete program

```
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("What is your first name? ");
        String name = console.next();

        System.out.print("And how old are you? ");
        int age = console.nextInt();

        System.out.println(name + " is " + age + ".  That's quite old!");
    }
}
```

Sample Run:

```
What is your first name? Marty
How old are you? 12
Marty is 12.  That's quite old!
```


Another complete program

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

public class Average {
    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();

        double average = (num1 + num2 + num3) / 3.0;
        System.out.println("The average is " + average);
    }
}
```

Sample Run:

```
Please type three numbers: 8 6 13
The average is 9.0
```

- Notice that the `Scanner` can read multiple values from one line.

Scanners as parameters

- The `main` method in the previous program could be better structured by grouping the collection of numbers into a method.

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

public class Average {
    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();

        double average = (num1 + num2 + num3) / 3.0;
        System.out.println("The average is " + average);
    }
}
```

Scanners as parameters

- To have multiple methods read user input, declare a `Scanner` in `main` and pass it to each method as a parameter.

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    double average = sum / 3.0;
    System.out.println("The average is " + average);
}

public static int readSum3(Scanner console) {
    System.out.print("Please type three numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```


Another complete program: Version 2

- Consider changing the output to include the minimum value:

```
Please type three numbers: 8 6 13
The average is 9.0
The minimum value is 6
```

- How would we change the previous program?

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    int sum = readSum3(console);
    double average = sum / 3.0;
    System.out.println("The average is " + average);
    // What goes here?
}

public static int readSum3(Scanner console) {
    System.out.print("Please type three numbers: ");
    int num1 = console.nextInt();
    int num2 = console.nextInt();
    int num3 = console.nextInt();
    return num1 + num2 + num3;
}
```


Methods cannot return more than one value!

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

public class Average {
    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();

        double average = (num1 + num2 + num3) / 3.0;
        System.out.println("The average is " + average);
        System.out.println("The minimum value is " +
                           Math.min(num1, Math.min(num2, num3)));
    }
}
```


Exercise: BMI

- A person's body mass index (BMI) is computed as follows:

$$BMI = \frac{weight}{height^2} \times 703$$

- Write a program that produces the following output:

```
This program reads in data for two people
and computes their body mass index (BMI)
and weight status.

Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Enter next person's information:
height (in inches)? 58.5
weight (in pounds)? 90

Person #1 body mass index = 23.485824
Person #2 body mass index = 18.487836949375414
Difference = 4.997987050624587
```


Solution: BMI

```
// This program computes two people's body mass index (BMI)
// and compares them.  The code uses parameters and returns.

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

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);
        double bmi1 = processPerson(console);
        double bmi2 = processPerson(console);
        outputResults(bmi1, bmi2);
    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads in data for two people");
        System.out.println("and computes their body mass index (BMI)");
        System.out.println("and weight status.");
        System.out.println();
    }

    // report overall results
    public static void outputResults(double bmi1, double bmi2) {
        System.out.println("Person #1 body mass index = " + bmi1);
        System.out.println("Person #2 body mass index = " + bmi2);
        double difference = Math.abs(bmi1 - bmi2);
        System.out.println("Difference = " + difference);
    }
```


Solution: BMI

```
    // reads information for one person, computes their BMI, and returns it
    public static double processPerson(Scanner console) {
        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();

        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();
        System.out.println();

        double bmi = getBMI(height, weight);
        return bmi;
    }

    // Computes a person's body mass index based on their height and weight
    // and returns the BMI as its result.
    public static double getBMI(double height, double weight) {
        double bmi = weight / (height * height) * 703;
        return bmi;
    }
}
```

Loop techniques

Readings: 4.1

Loop techniques

- Cumulative sum
- Fencepost loops

Adding many numbers

- Consider the following code:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int num1 = console.nextInt();

System.out.print("Type a number: ");
int num2 = console.nextInt();

System.out.print("Type a number: ");
int num3 = console.nextInt();

int sum = num1 + num2 + num3;

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

- Any ideas to improve the code?

Cumulative sum

- The variables `num1`, `num2`, and `num3` are unnecessary:

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int sum = console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

System.out.print("Type a number: ");
sum += console.nextInt();

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

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

Cumulative sum

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

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

System.out.println("The sum is " + sum); // sum out of scope
```


Cumulative sum loop

- A correct version:

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

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

User-guided cumulative sum

- The user's input can control the number of times the loop repeats:

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

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

Sample Run:

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


Cumulative sum: Exercise

- Write a program that reads input of the number of hours two employees have worked and displays each employee's total and the overall total hours.
  - The company doesn't pay overtime, so cap any day at 8 hours.

Sample Run:

```
Employee 1: How many days? 3
Hours? 6
Hours? 12
Hours? 5
Employee 1's total paid hours = 19

Employee 2: How many days? 2
Hours? 11
Hours? 6
Employee 2's total paid hours = 14

Total paid hours for both employees = 33
```


Cumulative sum: Solution

```
// 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.

import java.util.*;

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

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

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

    ...
```


Cumulative sum: Solution

```
    ...

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

        // 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();
            hours = Math.min(hours, 8);       // cap at 8 hours per day
            totalHours += hours;
        }

        System.out.println("Employee " + number + "'s total paid hours = "
                           + totalHours);
        System.out.println();
        return totalHours;
    }
}
```


Fencepost loops
"How do you build a fence?"

Readings: 4.1

The fencepost problem

- Problem: Write a static method named `printNumbers` that prints each number from 1 to a given maximum, separated by commas.
- Example:
  `printNumbers(5)`

  should print:
  `1, 2, 3, 4, 5`

A solution?

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

- Output from `printNumbers(5)`:

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


How about this?

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

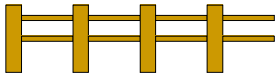
- Output from `printNumbers(5)`:

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


The fencepost problem

- We want to print *n* numbers but need only *n* - 1 commas.
- Similar to the task of building a fence
  - If we repeatedly place a post and wire, the last post has an extra dangling wire.
  - A flawed algorithm:

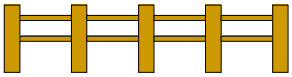
```
for (length of fence) {
    plant a post.
    attach some wire.
}
```



Fencepost loop

- The solution is to add an extra statement outside the loop that places the initial "post."
  - This is called a *fencepost loop*.
  - The revised algorithm:

```
plant a post.
for (length of fence - 1) {
    attach some wire.
    plant a post.
}
```



The fencepost solution

```
public static void printNumbers(int max) {
    System.out.print(1);
    for (int i = 2; i <= max; i++) {
        System.out.print(", " + i);
    }
    System.out.println();  // to end the line
}
```

- Output from `printNumbers(5)`:

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


Fencepost loop: Exercise

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

```
Base: 2
Max exponent: 9

The first 9 powers of 2 are:
2, 4, 8, 16, 32, 64, 128, 256, 512
```

`if/else` statements

Readings: 4.2

Conditionals

- "If you eat your vegetables, then you can have dessert."
- "If you do your homework, then you may go outside to play, or else you'll be grounded for life."

The `if` statement

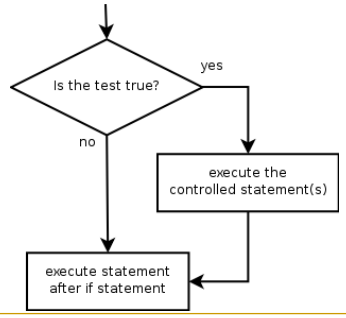
- **`if` statement**: A *control structure* that executes a block of statements only if a certain condition is true.
- General syntax:

```
if (<test>) {
    <statement(s)> ;
}
```

- Example:

```
double gpa = console.nextDouble();
if (gpa >= 3.0) {
    System.out.println("Good job! Have a cookie.");
}
```


`if` statement flow chart



The `if/else` statement

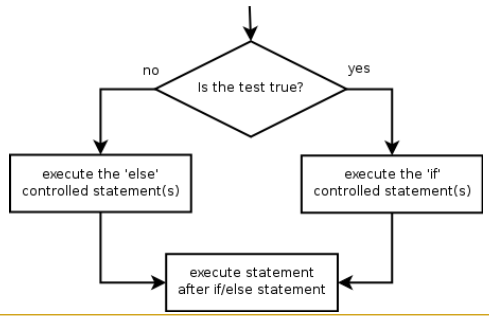
- **`if/else` statement**: A control structure that executes one block of statements if a certain condition is true, and a second block of statements if it is false. We refer to each block as a *branch*.
- General syntax:

```
if (<test>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```

- Example:

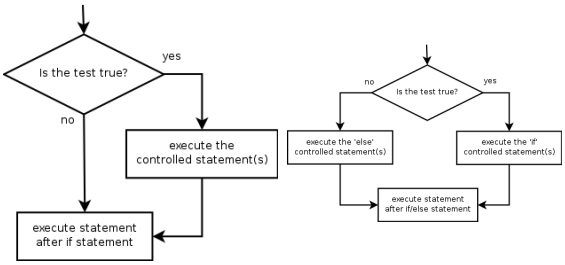
```
double gpa = console.nextDouble();
if (gpa >= 3.0) {
    System.out.println("Good job! Have a cookie.");
} else {
    System.out.println("No cookie for you!");
}
```


`if/else` statement flow chart



The non-existent loop

- There is no such thing as an “if loop”—there is no loop!



Relational expressions

- The ***<test>*** used in an `if` or `if/else` statement is the same kind seen in a `for` loop.

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

- These tests are called **relational expressions** and use the following **relational operators:**

Operator	Meaning	Example	Value
==	equals	1 + 1 == 2	true
!=	does not equal	3.2 != 2.5	true
<	less than	10 < 5	false
>	greater than	10 > 5	true
<=	less than or equal to	126 <= 100	false
>=	greater than or equal to	5.0 >= 5.0	true

Evaluating relational expressions

- Relational operators have lower precedence than math operators.

```
5 * 7 >= 3 + 5 * (7 - 1)
5 * 7 >= 3 + 5 * 6
35    >= 3 + 30
35    >= 33
true
```

- Relational operators cannot be "chained" as they can in algebra.

```
2 <= x <= 10
true   <= 10
   error!
```


`if/else`: Exercise

- Write code to read a number from the user and print whether it is even or odd using an `if/else` statement.

Example executions:

```
Type a number: 42
Your number is even

Type a number: 17
Your number is odd
```


Loops with `if/else`

- Loops can be used with `if/else` statements.

```
int nonnegatives = 0, negatives = 0;
for (int i = 1; i <= 10; i++) {
    int next = console.nextInt();
    if (next >= 0) {
        nonnegatives++;
    } else {
        negatives++;
    }
}
```

```
public static void printEvenOdd(int max) {
    for (int i = 1; i <= max; i++) {
        if (i % 2 == 0) {
            System.out.println(i + " is even");
        } else {
            System.out.println(i + " is odd");
        }
    }
}
```


Errors in coding

- Many students new to `if/else` write code like this:

```
int percent = console.nextInt();
if (percent >= 90) {
    System.out.println("You got an A!");
}
if (percent >= 80) {
    System.out.println("You got a B!");
}
if (percent >= 70) {
    System.out.println("You got a C!");
}
if (percent >= 60) {
    System.out.println("You got a D!");
} else {
    System.out.println("You got an F!");
}
```

- What’s the problem?

Nested `if/else` statements

- **Nested `if/else` statement**: A chain of `if/else` that can select between many different outcomes based on several tests.
- General syntax:

```
if (<test>) {
    <statement(s)> ;
} else if (<test>) {
    <statement(s)> ;
} else {
    <statement(s)> ;
}
```

- Example:

```
if (number > 0) {
    System.out.println("Positive");
} else if (number < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```


Nested `if/else` variations

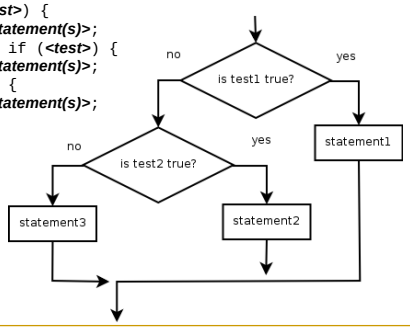
- A nested `if/else` can end with an `if` or an `else`.
  - If it ends with `else`, one of the branches must be taken.
  - If it ends with `if`, the program might not execute any branch.

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
```

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
}
```

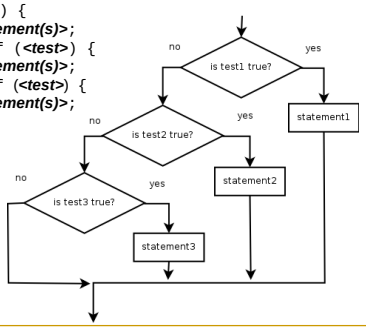

Nested `if/else` flow chart

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
```



Nested `if/else if` flow chart

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
}
```



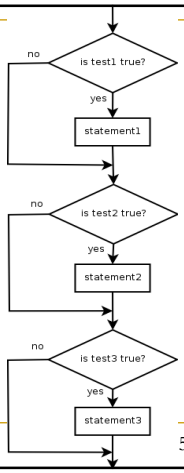
Nested `if/else` variations

```
if (place == 1) {
    System.out.println("You win the gold medal!");
} else if (place == 2) {
    System.out.println("You win a silver medal!");
} else if (place == 3) {
    System.out.println("You earned a bronze medal.");
}
```

- Are there any cases where this code will not print a message?
- How could we modify it to print a message to non-medalists?

Sequential `if` flow chart

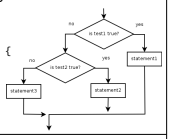
```
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
```



Summary: `if/else` structures

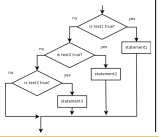
- Choose exactly 1 set of statements

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else {
    <statement(s)>;
}
```



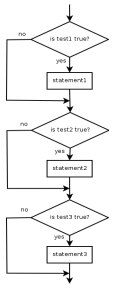
- Choose 0 or 1 set of statements

```
if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
} else if (<test>) {
    <statement(s)>;
}
```



- Choose 0, 1, or more set of statements

```
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
if (<test>) {
    <statement(s)>;
}
```



Which `if/else` construct to use?

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

Which `if/else` construct to use?

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

The `if/else` hammer

- Just because you learned a new construct does not mean that every new problem has to be solved using that construct!

```
int z;
if (x > y) {
    z = x;
} else {
    z = y;
}
```

```
int z = Math.max(x, y);
```

```
double d = a;
if (b < d) {
    d = b;
}
if (c < d) {
    d = c;
}
```

```
double d = Math.min(a, Math.min(b, c));
```


Factoring `if/else`

Readings: 4.3 (pg. 230 – 232)

Factoring `if/else`

- **factoring**: extracting common/redundant code
- Factoring `if/else` code reduces the size of the `if` and `else` statements
- Factoring tips:
  - If the start of each branch is the same, move it *before* the `if/else`.
  - If the end of each branch is the same, move it *after* the `if/else`.

Factoring: Before

```
if (money < 500) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Caution!  Bet carefully.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else if (money < 1000) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Consider betting moderately.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else {
    System.out.println("You have, $" + money + " left.");
    System.out.print("You may bet liberally.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
}
```


Factoring: After

```
System.out.println("You have, $" + money + " left.");

if (money < 500) {
    System.out.print("Caution!  Bet carefully.");
} else if (money < 1000) {
    System.out.print("Consider betting moderately.");
} else {
    System.out.print("You may bet liberally.");
}

System.out.print("How much do you want to bet? ");
bet = console.nextInt();
```


Subtleties of `if/else`

Readings: 4.3 (pg. 225 – 226)

Variable initialization

```
String message;
if (gpa >= 3.0) {
    message = "Welcome to the UW!";
}
if (gpa >= 2.0) {
    message = "Have you considered applying to WSU?";
}
if (gpa < 2.0) {
    message = "I hear Harvard still needs students...";
}
System.out.println(message);
```

- The compiler will complain that `"variable message might not have been initialized"`. Why?

Variable initialization

- The solution:

```
String message;
if (gpa >= 3.0) {
    message = "Welcome to the UW!";
} else if (gpa >= 2.0) {
    message = "Have you considered applying to WSU?";
} else { // gpa < 2.0
    message = "I hear Harvard still needs students...";
}
System.out.println(message);
```


Return

- Methods can return different values under different conditions:

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

public static String message(int place) {
    if (place == 1) {
        return "You won!";
    } else {
        return "If you're not first, you're last!";
    }
}
```

Errors in coding

```
public static int min(int a, int b) {
    if (a > b) {
        return b;
    }
}
```

- The compiler will complain about a `"missing return statement"`. Why?
- **ERROR**: Not returning a value in every path. In the above example, what if `a <= b`?

How about this?

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

- It still produces the `"missing return statement"` error. Why?
  - To our eyes, it is clear that all paths (greater, equal, less) do return a value.
  - But the compiler thinks that `if/else if` code might choose not to execute any branch, so it refuses to accept this code.
- How can we fix it?

Putting it all together: Exercises

- Write a method named `countFactors` that returns the number of factors of a given integer.
  - For example, `countFactors(60)` returns 12 because 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60 are factors of 60.
- Write a method named `numUnique` that accepts two integers as parameters and returns how many unique values were passed.
  - For example, `numUnique(3, 7)` returns 2 because 3 and 7 are two unique numbers, but `numUnique(4, 4)` returns 1 because 4 and 4 only represent one unique number.

Exercise: Counting primes

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

```
Maximum number? 50
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47
15 total primes
```


Solution: Counting primes

```
import java.util.*;

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

    public static int countFactors(int num) {
        int count = 0;
        for (int i = 1; i <= num; i++) {
            if (num % i == 0) {
                count++;
            }
        }

        return count;
    }

    ...
```


Solution: Counting primes

```
    public static int getNumber(Scanner console) {
        System.out.print("Maximum number? ");
        return console.nextInt();
    }

    public static void printPrimes(int max) {
        int numPrimes = 0;
        if (max >= 2) {
            System.out.print(2);
            numPrimes++;
            for (int i = 3; i <= max; i++) {
                if (countFactors(i) == 2) {
                    numPrimes++;
                    System.out.print(", " + i);
                }
            }
            System.out.println();
        }

        System.out.println(numPrimes + " total primes");
    }
}
```

Debugging 101

Readings: None

Why won't it toast?

- You arrive at your dorm after a thought-provoking lecture of CSE 142. To feed your brain, you put some bread into your toaster oven and set the dial for 5 minutes. The toaster oven ticks away. After five minutes, the toaster oven dings. You take the bread out, but it's not even toasted. What do you do?

What's wrong with this code?

```
import java.util.*;

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

        System.out.print("How many numbers to average? ");
        int count = console.nextInt();
        int sum = computeSum(console, count);
        System.out.println("The average is: " + (sum / count));
    }

    public static int computeSum(Scanner input, int num) {
        int total = 0;
        for (int i = 1; i <= num; i++) {
            System.out.print("#" + i + ": ");
            total = input.nextInt();
        }
        return total;
    }
}
```


Always remember

- Learn how to use the debugger
  - See the notes on the web page under "jGRASP Tutorial"
- `System.out.println` is your friend. Use it to print out variables and expressions.
  - Example:

```
System.out.println("x = " + x);
```