Java Programming Foundations 1

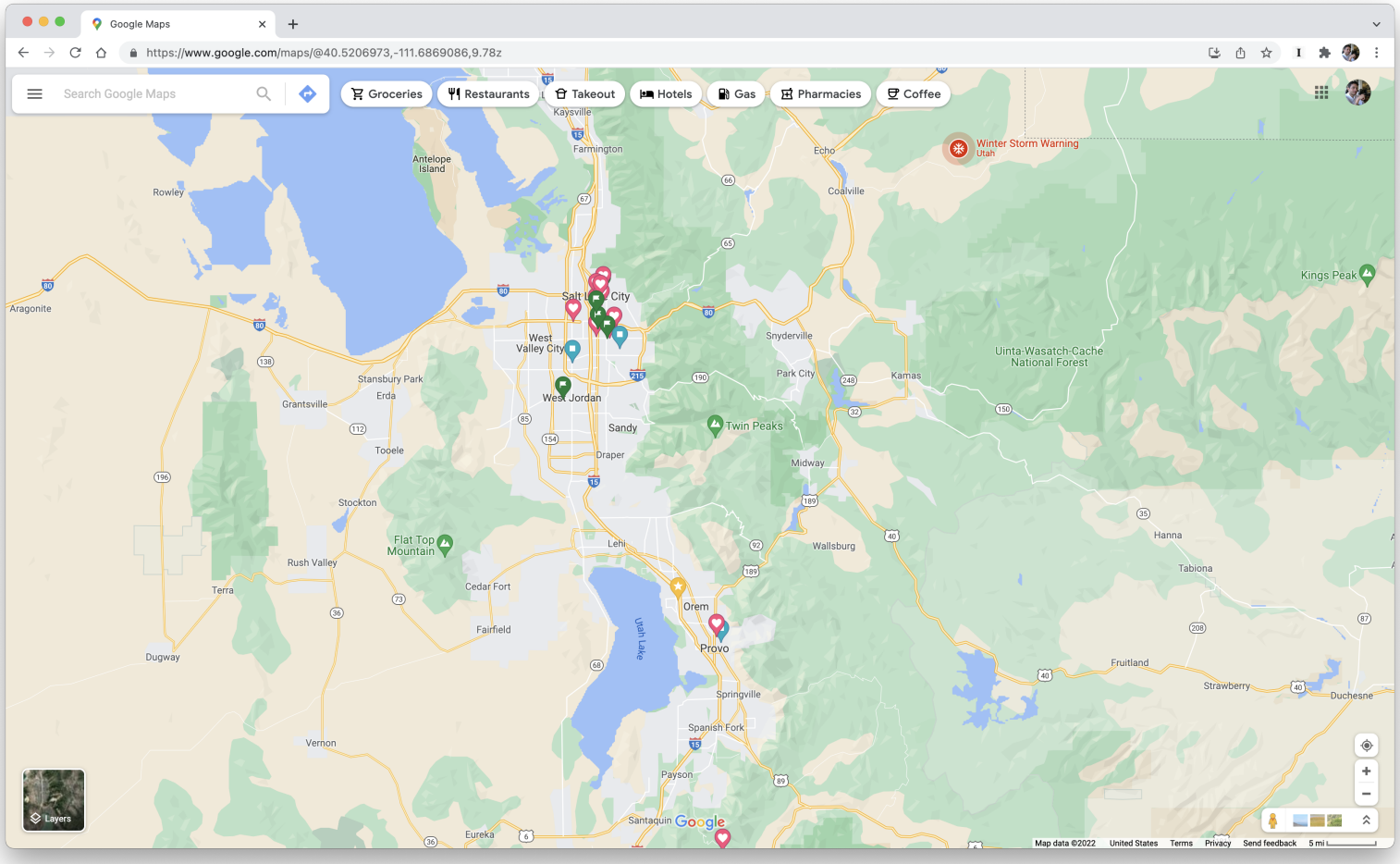
Week 7: Methods

Overview

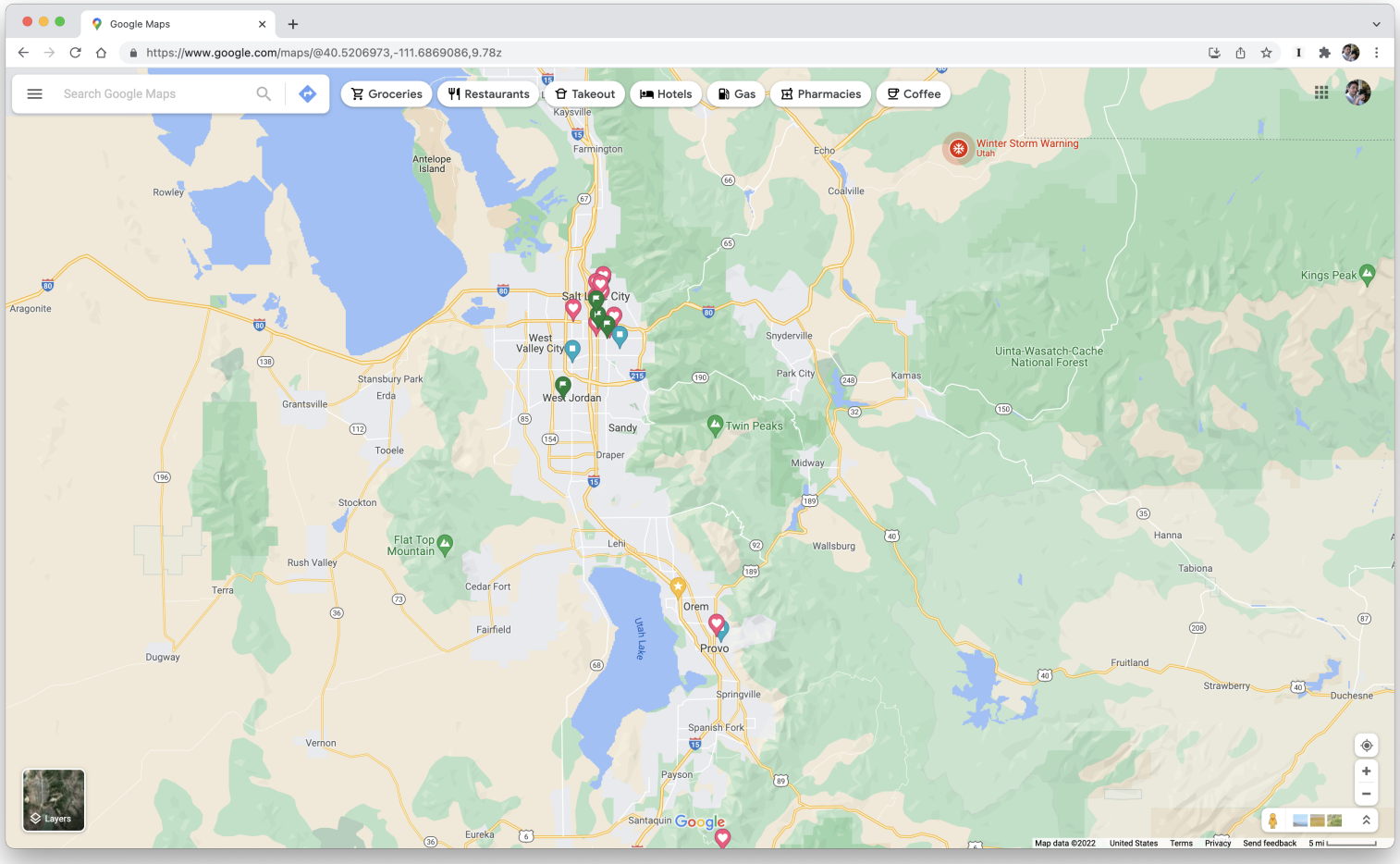
- A little bit of math with formulas
- A little bit of Java with methods
- Syntax
- Return results
- Scope
- Overloading
- Predefined methods

A little bit of math with formulas

Estimate how much time it will take to drive somewhere



How can we estimate how much time it will take to drive somewhere?



Formula to estimate how much time it will take to drive somewhere (1)

$$drive\ time\ in\ hours = distance\ in\ miles\ /\ average\ miles\ per\ hour$$

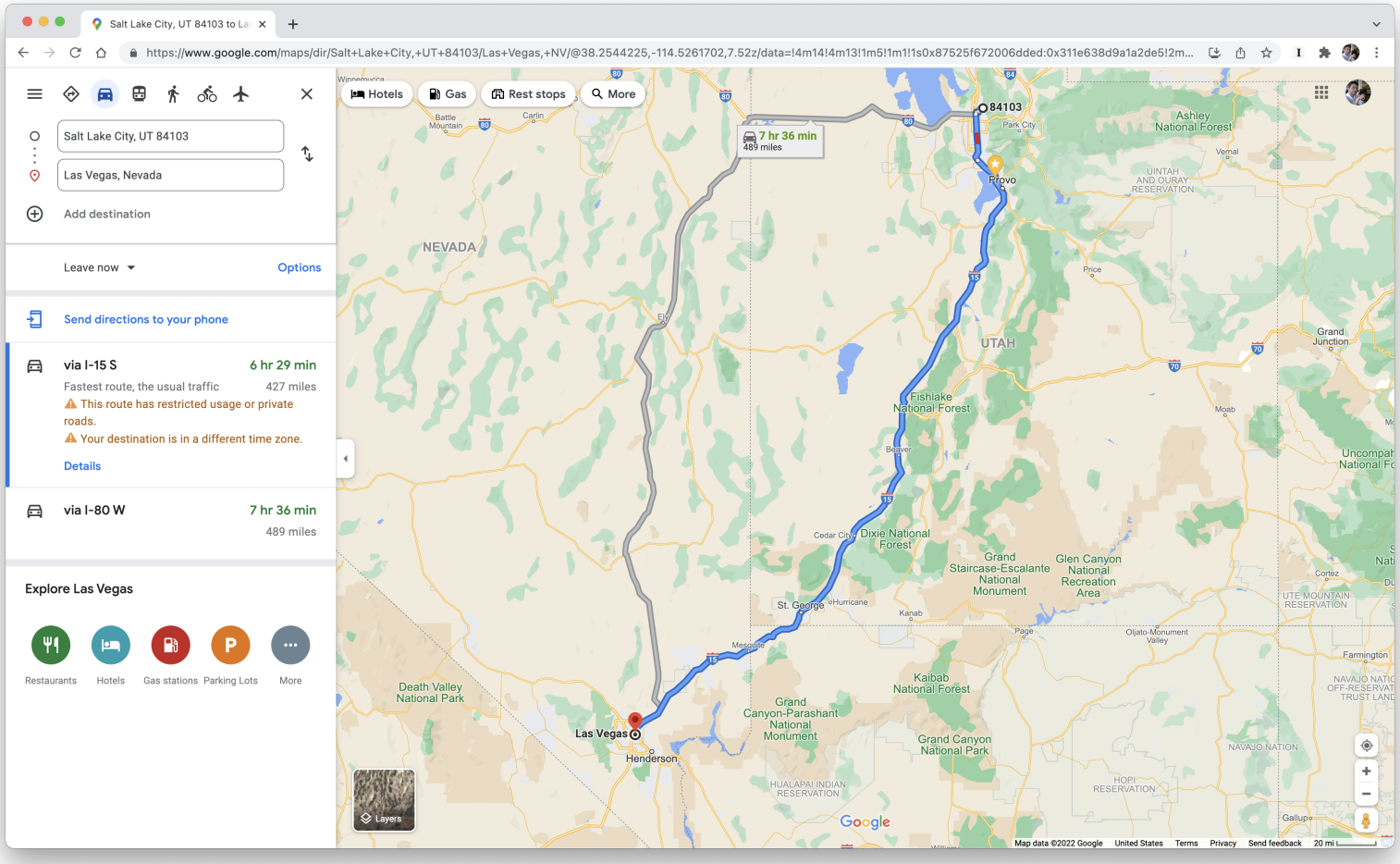
Formula to estimate how much time it will take to drive somewhere (2)

$$drive\ time\ in\ hours = \frac{distance\ in\ miles}{average\ miles\ per\ hour}$$

Formula to estimate how much time it will take to drive somewhere (3)

$$drive\ time = \frac{distance}{average\ mph}$$

Estimate how much time it will take to drive to Las Vegas



Estimate how much time it will take to drive to Las Vegas

$$distance = 427\ miles$$

$$average\ mph = 65\ mph$$

$$drive\ time = distance\ /\ average\ mph$$

Estimate how much time it will take to drive to Las Vegas

$$distance = 427\ miles$$
$$average\ mph = 65\ mph$$

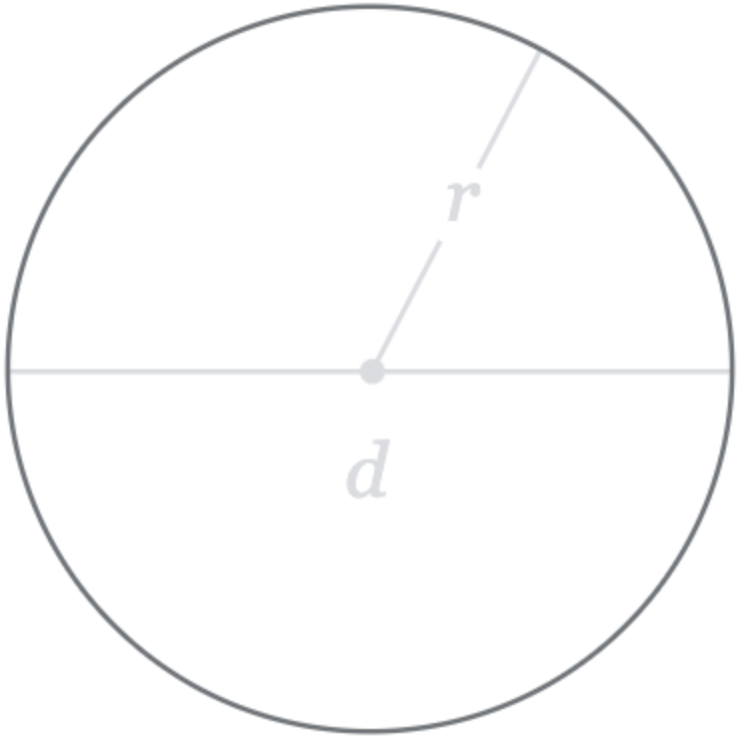
$$drive\ time = distance\ /\ average\ mph$$
$$= 427\ /\ 65$$

Estimate how much time it will take to drive to Las Vegas

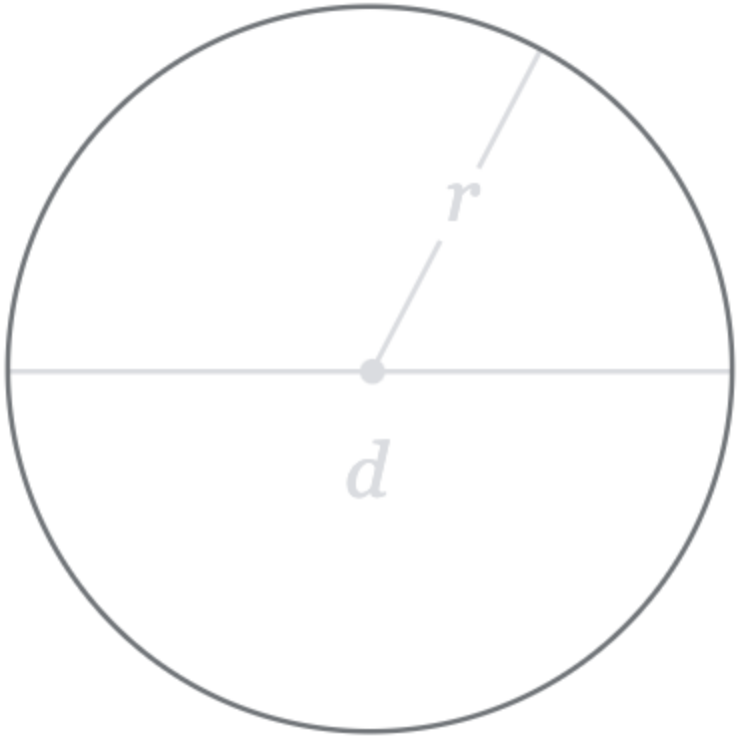
$$distance = 427\ miles$$
$$average\ mph = 65\ mph$$

$$\begin{aligned} drive\ time &= distance\ /\ average\ mph \\ &= 427\ /\ 65 \\ &\approx 6.6\ hours \end{aligned}$$

Calculate the area of a circle



How can we calculate the area of a circle?



Formula to calculate the area of a circle (1)

$$area = \pi \cdot radius^2$$

Formula to calculate the area of a circle (2)

$$area = \pi r^2$$

Calculate the area of a circle with a radius of 10 feet

$$radius = 10\ feet$$

$$area = \pi \cdot radius^2$$

Calculate the area of a circle with a radius of 10 feet

$$radius = 10\ feet$$

$$\begin{aligned} area &= \pi \cdot radius^2 \\ &= \pi \cdot 10^2 \end{aligned}$$

Calculate the area of a circle with a radius of 10 feet

$$radius = 10\ feet$$

$$\begin{aligned} area &= \pi \cdot radius^2 \\ &= \pi \cdot 10^2 \\ &= \pi \cdot 100 \end{aligned}$$

Calculate the area of a circle with a radius of 10 feet

$$radius = 10\ feet$$

$$\begin{aligned} area &= \pi \cdot radius^2 \\ &= \pi \cdot 10^2 \\ &= \pi \cdot 100 \\ &\approx 314.1 \end{aligned}$$

A little bit of Java with methods

Keep the following idea in your mind as we proceed

Methods allow you to have reusable code.

Keep the following idea in your mind as we proceed

Methods allow you to abstract your code.

Let's start with two samples

- Declaring methods
- Calling methods

Sample method declaration

```
public static void main(String[] args)
{
    // Method block, your code goes here
}
```

Sample method call

```
System.out.println("this is how to call a method in Java");
```

Syntax

The `main` method

```
              Return type
 Modifiers      |                Parameters
 |      \       |   Name           |
 |       \      |    |             |
 |        |     |    |             |
 V        V     V    V             V
public static void main(String[] args)
{
                                              <--+
                                                 |
                                                 |--- Method block
                                                 |
                                              <--+
}
```

Different name, multiple parameters

```
             Return type
 Modifiers      |                Parameters
 |      \       |   Name           |
 |       \      |     |            |
 |        |     |     |            |
 V        V     V     V            V
public static int anotherName(String name, int age)
{
                                              <--+
                                                 |
                                                 |--- Method block
                                                 |
                                              <--+
}
```

Let's write some code

Estimate how much time it will take to drive to Las Vegas

$$distance = 427\ miles$$
$$average\ mph = 65\ mph$$

$$\begin{aligned} drive\ time &= distance\ /\ average\ mph \\ &= 427\ /\ 65 \\ &\approx 6.6\ hours \end{aligned}$$

```java
public class CalculateEstimatedDriveTime
{
    public static void main(String[] args)
    {
        double distance = 427;
        double mph = 65;
        calcuateEstimatedDriveTime(distance, mph);
    }

    public static void calcuateEstimatedDriveTime(double distance, double mph)
    {
        double driveTime = distance / mph;
        System.out.println("Your estimated drive time is " + driveTime + " hours");
    }
}
```

Let's write some code

Birthday card message generator

"[name], congratulations on turning [age]. Here's to another year of being best friends!"

```
public class BirthdayCardMessageGenerator
{
    public static void main(String[] args)
    {
        String name = "Marcos";
        int age = 33;
        generateBirthdayCardMessage(name, age);
    }

    public static void generateBirthdayCardMessage(String name, int age)
    {
        System.out.println(name + ", congratulations on turning " + age +
          ". Here's to another year of being best friends!");
    }
}
```

How does this help me?

- Methods are reusable: you can easily call them multiple times
- Methods are parameterized: you can call them with different parameters (more on this later)

Reusable and parameterized

```
generateBirthdayCardMessage("Ryan", 40);
generateBirthdayCardMessage("Alec", 36);
generateBirthdayCardMessage("Sean", 37);
```

A life without methods

```
public class BirthdayCardMessageGenerator
{
    public static void main(String[] args)
    {
        String name1 = "Ryan";
        int age1 = 40;
        System.out.println(name1 + ", congratulations on turning " + age1 +
          ". Here's to another year of being best friends!");

        String name2 = "Alec";
        int age2 = 36;
        System.out.println(name2 + ", congratulations on turning " + age2 +
          ". Here's to another year of being best friends!");

        String name3 = "Sean";
        int age3 = 37;
        System.out.println(name3 + ", congratulations on turning " + age3 +
          ". Here's to another year of being best friends!");
    }
}
```

A life with methods

```
public class BirthdayCardMessageGenerator
{
    public static void main(String[] args)
    {
        generateBirthdayCardMessage("Ryan", 40);
        generateBirthdayCardMessage("Alec", 36);
        generateBirthdayCardMessage("Sean", 37);
    }

    public static void generateBirthdayCardMessage(String name, int age)
    {
        System.out.println(name + ", congratulations on turning " + age +
          ". Here's to another year of being best friends!");
    }
}
```

The syntax, again

```
              Return type
 Modifiers      |                Parameters
 |      \       |   Name           |
 |       \      |     |            |
 |        |     |     |            |
 V        V     V     V            V
public static int anotherName(String name, int age)
{
                                                <--+
                                                   |
                                                   |--- Method block
                                                   |
                                                <--+
}
```

The method block

```
public static int anotherName(String name, int age)
{
                                                <--+
                                                   |
                                                   |--- Method block
                                                   |
                                                <--+
}
```

The modifiers

```
 Modifiers
 |      \
 |       \
 |        |
 V        V
public static int anotherName(String name, int age)
{
}
```

The modifiers

There are additional modifiers other than `public` and `static`, but for today's class we will only use `public` and `static`.

The name

```
                      Name
                       |
                       |
                       V
public static int anotherName(String name, int age)
{
}
```

The name

Method names follow the same rules as variable names.

The name

A valid Java identifier is a “word” of arbitrary length composed of letters and/or digits and/or two special characters $ (dollar sign) and _ (underscore), where the first character must be a letter.

Java Programming: From the Ground Up, page 25

The parameters

```
                                    Parameters
                                       |
                                       |
                                       |
                                       V
public static int anotherName(String name, int age)
{
}
```

The parameters

- Parameters are optional
- Parameters must specify their "type"
- Parameters are only available in the method's block.
  More on this later

The return type

```
             Return type
                 |
                 |
                 |
                 |
                 V
public static int anotherName(String name, int age)
{
}
```

The return type

The return type specifies the "type" of the value that the method "returns" to the "caller".

The return type specifies the "type" of the value that the method "returns" to the "caller".

Returns: a method is able to "hand back" the result. This is referred to as "retuning" in Java. Think of this as the answer from the method. More on this coming.

Type: refers to the type of the value being returned by this method, This is the same as the type you've used in variables, such as `int`, `double`, `String`, etc.

Caller: the caller is the code that called the method. More on this coming.

Returning nothing

`void` means that this method does not return anything. `void` is the type which represents a lack of a value.

Returning nothing

```
public class CalculateEstimatedDriveTime
{
    /* ... main method omitted ... */

    public static void calcuateEstimatedDriveTime(double distance, double mph)
    {
        double driveTime = distance / mph;
        System.out.println("Your estimated drive time is " + driveTime + " hours");
    }
}
```

Returning a value

If instead we want to return a value, we need to specify the type of the value being returned and "return" it with the `return` keyword.

Estimate how much time it will take to drive to Las Vegas

```
public class CalculateEstimatedDriveTime
{
    /* ... main method omitted ... */

    /*
    public static void   calcuateEstimatedDriveTime(double distance, double mph)
    */
    public static double calcuateEstimatedDriveTime(double distance, double mph)
    {
        double driveTime = distance / mph;
        return driveTime;
    }
}
```

The `return` keyword

- Specifies the value that is returned to the caller
- Terminates the execution of the current method

Multiple `return` statements

```
public class CardMessageGenerator
{
    public static void main(String[] args)
    {
        String message = generateCardMessage("Ryan", 39, false);
        System.out.println(message);
    }

    public static String generateCardMessage(String name, int age, boolean isBirthday)
    {
        if (isBirthday) {
            return name + ", congratulations on turning " + age + "!";
        } else {
            return name + ", thanks for being such a good friend!";
        }
    }
}
```

The return type

Note that the return type specified in the method must match the type of the value being returned in the method block.

Think in terms of inputs and outputs

- The parameters are the inputs
- The returned value is the output

Caller

```
public class CalculateEstimatedDriveTime
{
    public static void main(String[] args)
    {
        double distance = 427;
        double mph = 65;
        double driveTime = calcuateEstimatedDriveTime(distance, mph);
        System.out.println("Your estimated drive time is " + driveTime + " hours");
    }

    public static double calcuateEstimatedDriveTime(double distance, double mph)
    {
        double driveTime = distance / mph;
        return driveTime;
    }
}
```

Let's write some code

Calculate the area of a circle with a radius of 10 feet

$$radius = 10\ feet$$

$$\begin{aligned} area &= \pi \cdot radius^2 \\ &= \pi \cdot 10^2 \\ &= \pi \cdot 100 \\ &\approx 314.1 \end{aligned}$$

```
public class AreaOfACircle
{
    public static void main(String[] args)
    {
        double radius = 10;
        double area = calculateAreaOfACircle(radius);
        System.out.println("The area of a circle with a radius of " + radius +
            " feet equals " + area + " squared feet");
    }

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

Scope and local variables

Sample of a local variable

```
public class LocalVariablesSample
{
    public static void main(String[] args)
    {
        String name = "Marcos";
        System.out.println(name);
    }

    public static void firstWithLocalVariables()
    {
        String message = "Hi, and welcome to Java!";
        System.out.println(message);
    }
}
```

Scope and local variables

Local variables that are declared in a method block are only able to be used inside of that method.

Scope and local variables

This is known as a variable's scope.

Local variables are only able to be used inside of the method block

```
public class LocalVariablesSample
{
    public static void firstWithLocalVariables()
    {
        String message = "Hi, and welcome to Java!";
        System.out.println(message);
    }

    public static void secondWithLocalVariables()
    {
        String favoriteNumber = 27;
        System.out.println(favoriteNumber);
    }
}
```

Same name, different method

```
public class LocalVariablesSample
{
    public static void firstWithLocalVariables()
    {
        String message = "Hi, and welcome to Java!";
        System.out.println(message);
    }

    public static void secondWithLocalVariables()
    {
        String message = "Have a great day!";
        System.out.println(message);
    }
}
```

Method parameters behave just like local variables

```
public class BirthdayCardMessageGenerator
{
    public static void main(String[] args)
    {
        String name = "Marcos";
        int age = 33;
        generateBirthdayCardMessage(name, age);
    }

    public static void generateBirthdayCardMessage(String name, int age)
    {
        System.out.println(name + ", congratulations on turning " + age +
          ". Here's to another year of being best friends!");
    }
}
```

Let's write some code

Write a method called `greaterThan10` that returns true when a number is greater than 10

```
greaterThan10(-1);   // Should return false
greaterThan10(3);    // Should return false
greaterThan10(10);   // Should return false
greaterThan10(43);   // Should return true
greaterThan10(6540); // Should return true
```

```
public class GreaterThanSample
{
    public static void main(String[] args)
    {
        System.out.println(greaterThan10(-1));
        System.out.println(greaterThan10(3));
        System.out.println(greaterThan10(10));
        System.out.println(greaterThan10(43));
        System.out.println(greaterThan10(6540));
    }

    public static boolean greaterThan10(int number)
    {
        return number > 10;
    }
}
```

Let's write some code

Write a method called `sum` that returns the sum of two numbers

```
sum(43, 43);     // 86
sum(3, 543);     // 546
sum(10, 1);      // 11
sum(43, 65);     // 108
sum(6540, 54);   // 6594
```

```
public class SumSample
{
    public static void main(String[] args)
    {
        System.out.println(sum(43, 43));
        System.out.println(sum(3, 543));
        System.out.println(sum(10, 1));
        System.out.println(sum(43, 65));
        System.out.println(sum(6540, 54));
    }

    public static int sum(int num1, int num2)
    {
        return num1 + num2;
    }
}
```

Overloading

Overloading?

Overloading, or method overloading, is the act of declaring multiple methods with the same name but different number of parameters or different parameter types.

Let's write some code

Card message generator

- "[name] thanks for being such a good friend!"
- "[name], congratulations on turning [age]!"

```java
public class CardMessageGenerator
{
    public static void main(String[] args)
    {
        System.out.println(generateCardMessage("Marcos"));
        // Marcos, thanks for being such a good friend!

        System.out.println(generateCardMessage("Marcos", 33));
        // Marcos, congratulations on turning 33!
    }

    public static String generateCardMessage(String name)
    {
        return name + ", thanks for being such a good friend!";
    }

    public static String generateCardMessage(String name, int age)
    {
        return name + ", congratulations on turning " + age + "!";
    }
}
```

Method overloading, let's zoom in

```
public class BirthdayCardMessageGenerator
{
    public static String generateCardMessage(String name) { /* ... */ }
    public static String generateCardMessage(String name, int age) { /* ... */ }
}
```

Method headers

A method is described by its header, which has the following form: `return-type name(parameter-list)`.

Java Programming: From the Ground Up, page 194

Method headers

- `String generateCardMessage(String name)`
- `String generateCardMessage(String name, int age)`

Limitations on overloading

Note that you cannot overload on the return type.

Predefined methods

Useful math methods in Java

(See next slide)

Java Programming: From the Ground Up, page 195

Figure 6.4 lists some useful methods found in the Math class. In each case, the first two columns comprise the header for each method.

Return Type	Method	Description	Example
double	abs(double x)	absolute value	Math.abs(−3.1) returns 3.1
int	abs(int a)	absolute value	Math.abs(−25) returns 25
double	ceil(double x)	returns the smallest whole number (as a double) greater than or equal to x	Math.ceil(3.14159) returns 4.0
double	cos(double x)	cosine function, x is in radians	Math.cos(3.141592653589793) returns −1.0 ($\cos(\pi) = -1$)
double	exp(double x)	the exponential function, e^x	Math.exp(0.0) returns 1.0 ($e^0 = 1$)
double	floor(double x)	returns the largest whole number (as a double) less than or equal to x	Math.floor(3.14159) returns 3.0
double	log(double x)	natural logarithm, $ln(x)$	Math.log(1.0) returns 0.0 ($\ln(1) = 0$)
double	max(double x, double y)	returns the greater of x and y	Math.max(3.0,4.0) returns 4.0
int	max(int a, int b)	returns the greater of x and y	Math.max(3,4) returns 4 (int)
double	min(double x, double y)	returns the lesser of x and y	Math.min(3.0,4.0) returns 3.0
int	min(int a, int b)	returns the lesser of a and b	Math.min(3,4) returns 3 (int)
double	pow(double x, double y)	x^y	Math.pow(2.0,5.0) returns 32.0
double	random()	returns a random number x such that 0.0 <= x < 1	Math.random() may return 0.2345676889 or perhaps 0.654678756
long	round(x double)	rounds to the nearest whole number (long)	Math.round(3.14) returns 3 (long) Math.round (5.67) returns 6 (long)
double	sin(double x)	sine function, x is in radians	Math.sin(3.141592653589793) returns 0.0 ($\sin(\pi) = 0$)
double	sqrt(double x)	square root	Math.sqrt(144.0) returns 12.0
double	tan(double x)	tangent function, x is in radians	Math.tan(3.141592653589793) returns 0.0 ($\tan(\pi) = 0$)

FIGURE 6.4 Methods of the *Math* class

Useful math methods in Java

```
public class MathMethodsSample
{
    public static void main(String[] args)
    {
        System.out.println( Math.min(123, 54) );     // 54
        System.out.println( Math.max(123, 54) );     // 123
        System.out.println( Math.pow(10, 2) );       // 100.0
        System.out.println( Math.random() );         // 0.5472084750229163
        System.out.println( Math.round(Math.PI) );   // 3
    }
}
```

Hint

You're going to need to use the `Math.sqrt` method to get the square root for one of the question in this week's homework.

Hint

When speaking about arithmetic, product means multiplication.