Learning Objectives - Static Methods

- **Define the term static method**
- **Identify when using a static method makes sense**
- **Demonstrate the syntax for calling an instance method**
- **Explain the relationship between static methods and**
- **Differentiate method scope between instance methods and static methods**

Static Methods

Static Methods

Imagine a `Rectangle` class in which objects have a length, width, and a method to calculate the area.

```
//add class definitions below this line

class Rectangle {
  int width;
  int length;

  Rectangle(int w, int l) {
    width = w;
    length = l;
  }

  int area() {
    return width * length;
  }
}

//add class definitions above this line
```

Create two instances of the `Rectangle` class, and then calculate the combined area of the two rectangles.

```
    //add code below this line

    Rectangle rect1 = new Rectangle(12, 27);
    Rectangle rect2 = new Rectangle(9, 3);
    int combinedArea = rect1.area() + rect2.area();
    System.out.println(combinedArea);

    //add code above this line
```

This works, but the combined area has to be calculated by the user. Since the combined area is related to the `Rectangle` class, a better solution would be to add this functionality to the class. Another type of method in Java is a

static method. Static methods use the `static` keyword in the method definition.

Static methods are most often used to add functionality to the whole class and not just an instance of the class. A good rule of thumb is to use a static method when none of the attributes of an instance are changed. In the example below, the attributes for the `r1` and `r2` object do not change. So a static method is a better choice than an instance method.

```
//add class definitions below this line

class Rectangle {
  int width;
  int length;

  Rectangle(int w, int l) {
    width = w;
    length = l;
  }

  int area() {
    return width * length;
  }

  static int combinedArea(Rectangle r1, Rectangle r2) {
    return r1.area() + r2.area();
  }
}

//add class definitions above this line
```

Static methods are called in a unique way. The still use dot notation, but instead of using the instance name use the class name followed by the dot and method name.

```
    //add code below this line

    Rectangle rect1 = new Rectangle(12, 27);
    Rectangle rect2 = new Rectangle(9, 3);
    System.out.println(Rectangle.combinedArea(rect1, rect2));

    //add code above this line
```

challenge

Try this variation:

Create the static method `describe` for the `Rectangle` class that prints a description of the rectangle.

▼ **Code**

```
static void describe(Rectangle r) {
  System.out.println("The rectangle has width of " +
      r.width + " and a length of " + r.length + ".");
}
```

Independence From Objects

Independence From Objects

Another unique characteristic of static methods is that they do not require the instantiation of an object before you can use them. Take a look at the `Car` class. It has some instance attributes, a constructor, and the static method `honk`.

```
//add class definitions below this line

class Car {
  String make;
  String model;
  String color;

  Car(String ma, String mo, String co) {
    make = ma;
    model = mo;
    color = co;
  }

  static void honk() {
    System.out.println("Beep! Beep!");
  }
}

//add class definitions above this line
```

In the `main` method, **do not** instantiate a `Car` object. Instead, call the `honk` method from the class.

```
    //add code below this line

    Car.honk();

    //add code above this line
```

This works because `honk` is a static method. If you remove the `static` keyword from the method definition, your program will not compile. You have used several static methods several times up to now. Type casting

between different data types is often done with static methods.

- `String.valueOf(20)` - This static method returns the string representation of integer 20.
- `Integer.parseInt("20")` - This static method returns the integer representation of the string “20”.
- `Double.valueOf(20)` - This static method returns the double representation of the integer 20.

challenge

Try this variation:

The `Math` class is full of static methods. Using the Java documentation for the `Math` class, see if you can write code using static methods that do the following things:

- Prints the largest number between `17` and `142`
- Prints the absolute value of `-2.34`
- Prints `3` to the power of `5`
- Prints the cosine of `34.1`
- Prints a random number between 0 and 1

▼ **Code**

Here are the static methods:

```
System.out.println(Math.max(17, 142));
System.out.println(Math.abs(-2.34));
System.out.println(Math.pow(3, 5));
System.out.println(Math.cos(34.1));
System.out.println(Math.random());
```

Limits of Static Methods

Because static methods are independent from objects, that means that static methods cannot directly access instance attributes of an object. Add the following static method to the `Car` class.

```
static void describe() {
  System.out.println(color + " " + make + " " + model);
}
```

Now call this method from the `Car` class as before. **Important**, running this code will cause an error. Java says that a non-static variable (instance attribute) cannot be referenced from a static context. That is, the static method `describe` cannot directly access the `color`, `make`, or `model` instance attributes of the `Car` class.

```
    //add code below this line

    Car.describe();

    //add code above this line
```

The way to avoid these types of errors is to pass a `Car` object to the `describe` method. Static methods can indirectly access instance attributes through an instance of the class. Change the `describe` method so that it has a `Car` object as a parameter. `describe` can now access the instance attributes by referencing the object name and the attribute using dot notation.

```
static void describe(Car c) {
  System.out.println(c.color + " " + c.make + " " + c.model);
}
```

Instantiate an instance of the `Car` class, and then pass this instance to the `describe` method. Java should run the code without an error.

```
    //add code below this line

    Car myCar = new Car("Honda", "Accord", "red");
    Car.describe(myCar);

    //add code above this line
```