Unit 7: Class Fields

After this unit, students should:

- understand the difference between instance fields and class fields
- understand the meaning of keywords `final` and `static` in the context of a field
- be able to define and use a class field
- be able to use `import` to access classes from the Java standard libraries

Class Fields

Let's revisit the following implementation of `Circle`.

```
// Circle v0.3
class Circle {
  private double x;
  private double y;
  private double r;

  public Circle(double x, double y, double r) {
    this.x = x;
    this.y = y;
    this.r = r;
  }

  public double getArea() {
    return 3.141592653589793 * this.r * this.r;
  }
}
```

In the code above, we use the constant π but hardcoded it as 3.141592653589793. Hardcoding such a magic number is a *no-no* in terms of coding style. This constant can appear in more than one places. If we hardcode such a number and want to change its precision later, we would need to trace down and change every occurrence. Every time we need to use π, we have to remember or look up what is the precision that we use. Not only does this practice introduce more work, it is also likely to introduce bugs.

In C, we define π as a macro constant `M_PI`. But how should we do this in Java? This is where the ideal that a program consists of only objects with internal states that communicate with each other can feel a bit constraining. The constant π is universal, and does not really belong to any object (the value of π is the same for every circle!).

Another example is the method `sqrt()` that computes the square root of a given number. `sqrt` is a general function that is not associated with any object as well.

A solution to this is to associate these *global* values and functions with a *class* instead of with an *object*. For instance. Java predefines a `java.lang.Math` class[1] that is populated with constants `PI` and `E` (for Euler's number *e*), along with a long list of mathematical functions. To associate a method or a field with a class in Java, we declare them with the `static` keyword. We can additionally add a keyword `final` to indicate that the value of the field will not change and `public` to indicate that the field is accessible from outside the class. In short, the combination of `public static final` modifiers is used for constant values in Java.

```java
class Math {
  :
  public static final double PI = 3.141592653589793;
  :
  :
}
```

We call these `static` fields that are associated with a class as *class fields*, and fields that are associated with an object as *instance fields*. Note that, a `static` class field needs not be `final` and it needs not be `public`. Class fields are useful for storing pre-computed values or configuration parameters associated with a class rather than individual objects.

Accessing Class Fields

A class field behaves just like a global variable and can be accessed in the code, anywhere the class can be accessed. Since a class field is associated with a class rather than an object, we access it through its *class name*. To use the static class field `PI`, for instance, we have to say `java.lang.Math.PI`.

```java
public double getArea() {
  return java.lang.Math.PI * this.r * this.r;
}
```

A more common way, however, is to use `import` statements at the top of the program. If we have this line:

```java
import java.lang.Math;
```

Then, we can save some typing and write:

```java
public double getArea() {
  return Math.PI * this.r * this.r;
}
```

Class Fields and Methods in Python

Note that, in Python, any variable declared within a `class` block is a class field:

```python
class Circle:
  x = 0
  y = 0
```

In the above example, `x` and `y` are class fields, not instance fields.

Example: The Circle class

Now, let revise our `Circle` class to improve the code and make it a little more complete. We now add in comments for each method and variable as well, as we always should.

```java
// version 0.4
import java.lang.Math;

/**
 * A Circle object encapsulates a circle on a 2D plane.
 */
class Circle {
  private double x;  // x-coordinate of the center
  private double y;  // y-coordinate of the center
  private double r;  // the length of the radius

  /**
   * Create a circle centered on (x, y) with given radius
   */
  public Circle(double x, double y, double r) {
    this.x = x;
    this.y = y;
    this.r = r;
  }

  /**
   * Return the area of the circle.
   */
  public double getArea() {
    return Math.PI * this.r * this.r;
  }

  /**
   * Return true if the given point (x, y) is within the circle.
   */
  public boolean contains(double x, double y) {
    return false;
    // TODO: Left as an exercise
    }
}
```

1. The class `Math` is provided by the package `java.lang` in Java. A package is simply a set of related classes (and interfaces, but I have not told you what is an interface yet). To use this class, we need to add the line `import java.lang.Math` at the beginning of our program. ↩