Fundamentals of Java

Defining a class Part 2

Method parameters

Until now, method parameters have been empty.

```
Car herbie;
herbie = new Car();

System.out.println(herbie.position);

herbie.drive();
herbie.drive();
herbie.drive();

System.out.println(herbie.position);
```

What could these parameters be used for?

Method parameters

e.g. Create car at position 100...

```
Car herbie;
herbie = new Car(100);

System.out.println(herbie.position);

herbie.drive(50);
herbie.drive(100);
herbie.drive(10);

System.out.println(herbie.position);
```

Where is the car now?

How must the definition of class Car change?

Definition of parameters

```
class Car {
    int position;

    Car(int position) {
        this.position = position;
    }
    void drive(int distance) {
        position = position + distance;
    }
}
```

Note the line: `this.position = position;`

`position` refers to the parameter

`this.position` refers to the field

Arguments vs. Parameters

Technically...

An argument is the value passed to a parameter.

```
herbie.drive(50);
```
<- 50 is an argument

```
void drive(int distance) {
    this.position = this.position + distance;
}
```
<- distance is a parameter

Example

```
Person harry = new Person("Harry Burns", 31);
Person sally = new Person("Sally Albright", 28);

harry.sayHello();
harry.grow(3);
harry.sayHello();
sally.sayHelloTo(harry);
```

Example

```
class Person {
    String name;
    int age;
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    void grow(int amount) {
        this.age = this.age + amount;
    }
    void sayHello() {
        System.out.println("Hi, my name is " + this.name + ".");
        System.out.println("I'm " + this.age + " years old.");
    }
    void sayHelloTo(Person other) {
        System.out.println("Hi, " + other.name + ".");
        System.out.println("I'm " + this.name + ". Nice to meet you!");
    }
}
```

Functional programming

A function is a method that returns a result.

e.g. Class Math provides many familiar mathematical functions:

```
sin(x), cos(x), sqrt(x), pow(x,y)...
```

Using a function:

```
double x = Math.sqrt(16.0);
```

A function does not print its result, it *returns* it.

Functional programming

A function call produces a value and therefore qualifies as an expression.

i.e. A function call can be used anywhere that a value is required:

```
double z = Math.sqrt(Math.pow(x2-x1, 2)
                   + Math.pow(y2-y1, 2));
```

Example

```
Triangle t1 = new Triangle(10.0, 4.0);

double area = t1.area();

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

Example

```
class Triangle {
    double base;
    double height;
    Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }
    double area() {
        double result;
        result = this.base/2.0 * this.height;
        return result;
    }
}
```

Keyword "this"

Java's "this" keyword is mostly optional.

Only absolutely necessary to avoid ambiguity.

```
class Circle {
    double radius;
    Circle(double radius) {
        this.radius = radius;
    }
    double area() {
        return Math.PI * radius * radius;
    }
}
```

this *necessary here*

this *not necessary here*

public and private

A programmer should declare whether each method or field of a class is public or private.

- A "public" method/field can be accessed by any other object from another class.
- A "private method/field can only be accessed by objects within the same class.

By default, the method/field is accessible only by other classes in the same package.

The rule of "encapsulation"

In object-oriented software design, fields should always be declared private.

Thus, the only way to interact with an object should be via its methods, not its fields.

The designer of a class can then control, via its methods, how other objects may interact with the objects in this class.

Encapsulating class Car

```
public class Car {
    private int position;

    public void drive(int distance) {
        position = position + distance;
    }
}
```

Accessing private data

```
Car herbie;
herbie = new Car();
herbie.position = 0;

System.out.println(herbie.position);

herbie.drive(100);
herbie.drive(250);

System.out.println(herbie.position);
```

Access to **herbie.position** is now denied.

Accessing private data

To access private data, we need to define public methods that provide access to the private data.

```
System.out.println(herbie.position);
```

becomes

```
System.out.println(herbie.getPosition());
```

and

```
herbie.position = 0;
```

becomes

```
herbie.setPosition(0);
```

Accessor methods

```
public class Car {
    private int position;
    public void drive(int distance) {
        position = position + distance;
    }
    public int getPosition() {
        return position;
    }
    public void setPosition(int position) {
        this.position = position;
    }
}
```

Accessor methods

Accessor methods provide more control than simply making fields public.

We can selectively decide to:

- Provide `getPosition()` access.
- Not provide `setPosition()` access
  i.e. no teleportation!

Composition revisited

Let's add public/private, parameters and functions to our compositional example.

```
public class Motorbike {
    private Wheel frontWheel;
    private Wheel backWheel;
    ...
}
```

A Motorbike is *composed of* two Wheels.

Composition

A Wheel has a position:

```
public class Wheel {
    private double position;
    ...
}
```

Constructor parameter

```
public class Motorbike {
    private Wheel frontWheel;
    private Wheel backWheel;
    public Motorbike(int position) {
        frontWheel = new Wheel(position + 50);
        backWheel = new Wheel(position - 50);
    }
}
```

If you create a new Motorbike, this in turn creates two new Wheels.

Method composition

```
public class Motorbike {
    ...
    public void drive(double distance) {
        frontWheel.roll(distance);
        backWheel.roll(distance);
    }
}
```

If you drive a Motorbike, this will in turn roll the two wheels.

Information hiding

Does a motorbike has a position?

```
public class Motorbike {
    private Wheel frontWheel;
    private Wheel backWheel;
}
public class Wheel {
    private int position;
}
```

No, but we can make it "seem" like it does.

Encapsulation / Information hiding

```
public class Wheel {
    private int position;
    public int getPosition() {
        return position;
    }
}
```

An outsider knows only of the `getPosition()` method, and not of the `position` field.

```
Wheel w = new Wheel();
println("The wheel is at position: " + w.getPosition());
```

The internal details are encapsulated by the object.
The internal structure of the object is "hidden".

Encapsulation / Information hiding

```
public class Motorbike {
    private Wheel frontWheel;
    private Wheel backWheel;
    public int getPosition() {
        return frontWheel.getPosition() + backWheel.
getPosition();
    }
}
```

An outsider knows only of the `getPosition()` method, not how it works under the hood.

```
Motorbike b = new Motorbike();
println("The bike is at position: " + b.getPosition());
```

The internal structure of the bike is hidden.

Exercises

Read the following specification:

In the game of blackjack, the dealer will deal you and him/herself 2 cards. Your goal is to ask for more cards, one at a time, trying to get as close to a total value of 21 without going over. The dealer does the same. Whoever is closest wins. In a draw, the dealer wins. Typically, the dealer will play simultaneously against several players, using the same hand.

Exercises

This program involves 5 classes: Dealer, Player, Deck, Hand, Card.

Split into groups of students and try to write code for each class.

Exercises

Design a Dog compositionally using the following classes: Dog, Leg, Head.

Implement the following methods:

- print() - prints all details about the dog
- walk() - makes the dog take one step forward
- pickUp("Bone") - picks up an object
- drop() - drops a held object
- bark - makes a "Rufff!" sound