Fundamentals of Java

Defining a class Part 1

Defining a class - Part 1

When defining a class of objects, we must ask two questions:

1. What do objects of this class **have**?
2. What can objects of this class **do**?

Each thing an object has is called a **field**.

Each thing an object can do is called a **method**.

Example #1
Inferring the structure of an object

What does herbie have? (i.e. fields)

What can herbie do? (i.e. methods)

```
Car herbie;
herbie = new Car();
herbie.position = 0;
System.out.println(herbie.position);
herbie.drive();
herbie.drive();
herbie.drive();
System.out.println(herbie.position);
```

Determine what is inside herbie by looking for snippets of code of the form: `herbie.xyz`

Example #1
Defining the class

Each Car object has a position. (field)

Each Car object can drive. (method)

```
class Car {
   int position;

   void drive() {
      ...insert code here...
      ...how does this method work?...
   }
}
```

A field is *just a variable* which can store data.

A method typically contains code to act on a field.

Fields are not ordinary variables!

Fields are variables that belong to objects. e.g.

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

Each car has ***its own*** position field.

To refer to a field, you must prefix it with the object name + dot. e.g.

```
kitt.position
```

Example #1
Defining the class

Let's return to the definition of our Car class:

```
class Car {
   int position;

   void drive() {
      ...insert code here...
      ...how does this method work?...
   }
}
```

A method typically contains code to act on a field.

We want to set the position field...

Example #1
Defining the class

Let's return to the definition of our Car class:

```
class Car {
   int position;

   void drive() {
      this.position = ... what goes here?
   }
}
```

"this" is a special object reference meaning "me" or "myself". this.position therefore means "my position".

Example #1
Defining the class

Each Car object has a position. (field)

Each Car object can drive. (method)

```
class Car {
   int position;

   void drive() {
      this.position = this.position + 1;
   }
}
```

Example #2

What does jack have?

What can jack do?

```
Person jack = new Person();
jack.name = "Jack Reacher";
jack.age = 35;

jack.sayHello();
jack.grow();
jack.grow();
jack.sayHello();
```

Example #2

Each Person has a name and age.

Each Person can say hello and grow.

```
class Person {
    String name;
    int age;

    void sayHello() {
        System.out.println("Hi, my name is " + this.name +
".");
    }
    void grow() {
        this.age = this.age + 1;
        System.out.println("I am " + this.age + " years
old.");
    }
}
```

Constructors

A class may define a constructor method.

This method:

- must have the same name as the class
- has no "void" before it
- is invoked whenever a new object is created

A default "empty" constructor is provided if you don't define one.

Constructors

```
class Car {
    int position;

    Car() {
        this.position = 0;
    }

    void drive() {
        this.position = this.position + 1;
    }
}
```

Default values

If a field is not explicitly initialised, defaults are chosen:

Type	Default value
byte	(byte)0
short	(short)0
int	**0**
long	0L
float	0.0f
double	**0.0**
char	**'\0'**
boolean	**false**
Object	**null**

Default values

Fields get default values, local variables do not!

```
class Circle {
    double radius;
    void showDetails() {
        System.out.println(this.radius);   Prints 0.0
        double area;
        System.out.println(area);          Error
    }
}
```

Compiler error: Local variable "area" might not have been initialised.

Composition

An object may be composed of other objects:

```
class Motorbike {
  Wheel frontWheel;
  Wheel backWheel;
  Motorbike() {
    this.frontWheel = new Wheel();
    this.backWheel = new Wheel();
    this.frontWheel.roll();
    this.frontWheel.roll();
  }
  void drive() {
    this.frontWheel.roll();
    this.backWheel.roll();
  }
}
```

```
class Wheel {
  int position;

  Wheel() {
    this.position = 0;
  }

  void roll() {
    this.position = this.position +
1;
  }
}
```

NullPointerException

Be careful to initialise Object fields before you use them!

```
class Motorbike {
  Wheel frontWheel;
  Wheel backWheel;
  Motorbike() {
    this.frontWheel.roll();
    this.frontWheel.roll();
  }
  void drive() {
    this.frontWheel.roll();
    this.backWheel.roll();
  }
}
```

Defaults to null

Error: NullPointerException (frontWheel is null)

Static fields and methods

Keyword "static" designates a field or method shared by all instances of a class.

```
class BankAccount {
    String name;
    double balance;
    static double interestRate;

    void deposit(double amount) {
        this.balance = this.balance + amount;
    }
    static void increaseInterestRate(double amount) {
        this.interestRate = this.interestRate + amount;
    }
}
```

Static fields and methods

```
Account ryansAccount = new Account();
Account samsAccount = new Account();
```

Normal fields/methods are accessed on an instance of a class:

```
System.out.println("Ryan has $" + ryansAccount.balance);
System.out.println("Sam has $" + samsAccount.balance);
ryansAccount.deposit(100.0);
System.out.println("Now, Ryan has $" + ryansAccount.balance);
```

Static fields/methods are accessed on the class:

```
Account.increaseInterestRate(0.03);
System.out.println("THE interest rate is " + Account.interestRate);
```

Exercises

Consider the following program:

```
Rectangle rect = new Rectangle();
rect.width = 5.0;
rect.height = 4.0;
rect.showDimensions();
rect.shrink();
rect.showDimensions();
rect.showArea();
```

What does a rectangle have? What can it do?

Define class Rectangle with appropriate fields and methods so that the above code compiles.

Exercises

Define class Circle so that this program compiles:

```
Circle c1 = new Circle();
c1.radius = 3.0;
c1.showRadius();
c1.showDiameter();
c1.showArea();
c1.showCircumference();
```

Use Google to look up the formulas :)