Classes and Objects

Java Developer

StayAhead Training; April, 2022

Objects

Classes and Objects

- An <u>object</u> is a group of related state and/or behaviours
- State is data that can change over time
- Behaviour is encapsulated in one or more methods
- Object oriented languages, like Java, require us to think about the world in terms of objects - every *thing* is an object
- E.g. a library book is an object - it has state (ISBN, title, author, on loan) and behaviours (check out, check in)
- Some objects comprise mostly state, some mostly behaviour

Classes

Classes and Objects

- A class is a template for creating objects/a classification/a data type
- The class specifies what state and behaviours each object should have
- An object is AKA an instance of a class
- A LibraryBook class might specify that each instance has an ISBN, a title, and an author, but would not specify the values of those things
- A LibraryBook class might specify that each instance can be checked out & in
- Like the columns in a DB table, a class specifies the shape of the data

Classes

Classes and Objects

- Typical class layout:

```
class Book {
    // fields
    // constructor(s)
    // methods
}
```

Fields

Classes and Objects

- A <u>field</u> is the specification of an item of state
- It comprises a data type, a name, and an optional initial value, e.g.

```
boolean onLoan = false;
```

- Each class can have zero or many fields
- The fields ought to be related to one another
- If no value is assigned then the compiler will assign a default value, e.g. 0 for numeric types, false for boolean, null for Strings etc.

Fields

Classes and Objects

```
class Book {
  int isbn;
  String title;
  String author;
  boolean onLoan = false;
}
```

```
class App {
  public static void main() {
    var book1 = new Book();
    book1.title = "My Book";
  }
}
```

Methods

Classes and Objects

- A method is a specification of some behaviour
- It comprises a return (output) type, a name, zero or more parameters (input), and one or more instructions inside a code block, e.g.

```
void checkout() {
    // instructions
}
```

- Each class can have zero or many methods
- Each method ought to operate on one or more of the class's fields

Methods

Classes and Objects

```
class Book {
  // method declaration
  void checkout() {
    onLoan = true;
  }
}
```

```
class App {
  public static void main() {
    var book1 = new Book();
    // method invocation/call
    book1.checkout();
}
```

Method Input

Classes and Objects

- A method may accept some input data that it needs to do its work
- Method input, as specified in the method declaration, is a <u>parameter</u>, e.g.

```java
void setTitle(String title) {
    // instructions
}
```

- Method input, as specified in the method invocation/call, is an <u>argument</u>, e.g.

```java
book1.setTitle("Your Book");
```

- Two or more parameters and/or arguments must be comma separated

Method Input

Classes and Objects

```
class Book {
  // isbn is the parameter
  void setIsbn(int isbn) {
    this.isbn = isbn;
  }
}
```

```
class App {
  public static void main() {
    var book1 = new Book();
    // 1234 is the argument
    book1.setIsbn(1234);
}
```

Method Output

Classes and Objects

- A method may return something to the caller/produce some output, e.g.

```
 return something;
```

- Control passes back to the caller immediately
- If the method does not return something then the return type must be void

Method Output

Classes and Objects

```
class Book {
  int getIsbn() {
    return isbn;
  }
}
```

```
class App {
  public static void main() {
    var book1 = new Book();
    var n = book1.getIsbn();
}
```

Method Overloading*

Classes and Objects

- Method overloading is the presence of two or more methods in a class with the same name but with different parameter lists, e.g.

```
void checkout() {
  // TODO
}

void checkout(int numDays) {
  // TODO
}
```

- NB: the return type is irrelevant with regards overloading

Constructors

Classes and Objects

- A constructor is like a method and is called with the new keyword to instantiate the class/create an object of the class
- Its name must match the class name
- It must not return anything or specify a return type
- If you do **not** add one to your class then the compiler will add one for you with no parameters (a no-args constructor)
- If you do add one to your class then the compiler will **not** add one for you
- Constructors, like methods, may be overloaded

Constructors

Classes and Objects

- Constructors are used to control the way the class is instantiated
- E.g. if a library book must have a title and author, then the constructor can be used to ensure those items of state are provided
- The constructor of the class is what is called when you encounter the new keyword, e.g. `var book = new Book();`

Constructors

Classes and Objects

```
class Book {
  Book(String title, String author) {
    this.title = title;
    this.author = author;
  }
}
```

Constructors

Classes and Objects

```java
class App {
  public static void main(String[] args) {
    var book1 = new Book("My Book", "Stuart");
  }
}
```

The this Keyword*

Classes and Objects

- In a class the this keyword references the current object; consider…

```
var book1 = new Book("My Book", "Stuart");
```

- The new keyword results in the creation of an empty object

- The empty object is referenced in the constructor using this, e.g.

```
this.title = title;
```

- The code in a class operates on some object that does not exist until runtime; the this keyword is used to reference that object

Class Instantiation

Classes and Objects

- To instantiate a class is to create an object, e.g.

```
var book1 = new Book("My Book", "Stuart");
```

- The new keyword is followed by a constructor call
- The variable, book1, contains a reference to the newly created object
- Some classes are instantiated many times, others only once

Dot Notation

Classes and Objects

- Each of an object's members (fields and methods) is accessed via the dot notation (object.member), e.g.

```
var book1 = new Book("My Book", "Stuart");

var title = book1.title;

book1.checkout();
```

- Note the difference between the field (title) and the method (checkout)