Chapter 10: Defining and Using Classes

Learning Outcomes:

- ✓ Identify the importance, and uses, of defining custom data types
- ✓ Implement a custom data type into a program
- ✓ Use an array list, with objects from a custom class, to complete an interactive program

Prerequisite Knowledge:

- ✓ Complete Chapter 9
- ✓ Be able to instantiate objects from a class and call their respective methods/functions
- ✓ Be able to code independent methods and functions (with parameters) in a program

Keywords: Object Properties, Constructor, Instantiate, FOR EACH Loop

10.1 The Theory: Defining Classes

Char, integer, double and Boolean are all primitive data types that were used in former chapters; as useful as these data types are, sometimes, however, they are just not enough. Objects in the 'real world' cannot be described with a single, primitive data type. For example, a book cannot be captured as a single string, because a book has many characteristics, such as author, title and ISBN number. Java, like many modern languages, is 'object-orientated', enabling programmers to define their own data types that closely capture the details of real-life objects – such as books!

Creating a Data Type

Defining a new data type is easy: create a new class and then add the object properties. An object property should be set to either public or private; for now, set properties to public so they are visible and accessible to other classes. After the keyword [`public`], add the data type and the property name. Finally, compile the class so that it is ready for use in a program. Below is an example of how the new `Book` class can be utilised in a program. Notice that once an object has been created from the `Book` class, the object's properties can be set using the assignment operator [`=`].

```
public class Book{

        public String title;
        public String author;
        public String isbn;
}
```

```
public class TestRun{

        public static void main(String[] args){
                Book x = new Book(); //creates a new book object
                Book y = new Book();
                Book z = new Book();

                x.title = "The Hobbit"; //sets the properties of each book object
                x.author = "Tolkien, J. R. R.";
                x.isbn = "0-261-10221-4";

                y.title = "Harry Potter and the Prisoner of Azkaban";
                y.author = "Rowling, J. K.";
                y.isbn = "978-1594130021";

                z.title = "King, S.";
                z.author = "Misery";
                z.isbn = "978-1444741292";
        }
}
```

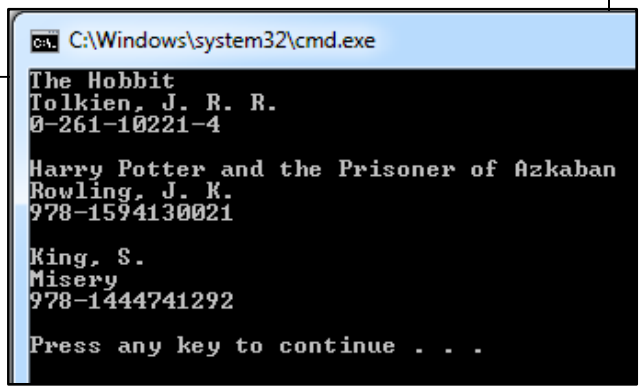

The Guitar Class

Once the properties of an object have been set, they can be referred to and used in program code. The syntax below demonstrates how object properties, once set, can be printed to the command-line:

Try the code

```
public class TestRun{

	public static void main(String[] args){
		Book x = new Book();
		Book y = new Book();
		Book z = new Book();

		x.title = "The Hobbit";
		x.author = "Tolkien, J. R. R.";
		x.isbn = "0-261-10221-4\n";

		y.title = "Harry Potter and the Prisoner of Azkaban";
		y.author = "Rowling, J. K.";
		y.isbn = "978-1594130021\n";

		z.title = "King, S.";
		z.author = "Misery";
		z.isbn = "978-1444741292\n";

		System.out.println(x.title); //print object properties to screen
		System.out.println(x.author);
		System.out.println(x.isbn);

		System.out.println(y.title);
		System.out.println(y.author);
		System.out.println(y.isbn);

		System.out.println(z.title);
		System.out.println(z.author);
		System.out.println(z.isbn);
	}
}
```

```
public class Book{

	public String title;
	public String author;
	public String isbn;
}
```



Activity 10.1

Write a `Guitar` class that holds the properties of a generic guitar. Properties may include guitar manufacturer, type, number of strings and number of frets.

Once complete, compile the `Guitar` class and use it in a program to generate three guitar objects. Set the properties of each guitar and then print all of the properties of each object to the command-line.

The Class and the Array

Arrays, used in conjunction with loops, are extremely useful and reduce the amount of code that would otherwise need to be written. Arrays and loops can also be implemented with objects, reducing the amount of code written when working with objects. In the example below, an array list is instantiated of type `Book` (resulting in the array list only accepting objects derived from the `Book` class); in the syntax below, the book objects `x`, `y` and `z` are added to the array list.

Another iteration (loop) type is introduced in the example below, the 'FOR EACH' loop [`for(Book myItem : myBooks)`]. The best way to understand this loop type is to think of it as 'FOR EACH `Book` type object found in the `myBooks` list'. For every iteration of the FOR EACH loop, the `Book` type object is stored into the temporary `myItem` variable; once stored into the `myItem` variable, the properties of the object (instantiated from the `Book` class) can be referenced to on each iteration, i.e. [`myItem.title`]. In other words, the FOR EACH loop below will iterate three times because there are three objects in the `myBooks` list; the `myItem` variable will hold object `x` on the first iteration, object `y` on the second and object `z` on the third.

Try the code

```
import java.util.ArrayList;
public class TestRun{

	public static void main(String[] args){
		ArrayList<Book> myBooks = new ArrayList<Book>(); //list of type book
		Book x = new Book();
		Book y = new Book();
		Book z = new Book();

		x.title = "The Hobbit";
		x.author = "Tolkien, J. R. R.";
		x.isbn = "0-261-10221-4 ";

		y.title = "Harry Potter and the Prisoner of Azkaban";
		y.author = "Rowling, J. K.";
		y.isbn = "978-1594130021";

		z.title = "King, S.";
		z.author = "Misery";
		z.isbn = "978-1444741292";

		myBooks.add(x); //add the book objects to the array list
		myBooks.add(y);
		myBooks.add(z);

		for(Book myItem : myBooks){ //for each book found in myBooks
			System.out.println("\n---->Books:");
			System.out.println("Title: " + myItem.title);
			System.out.println("Author: " + myItem.author);
			System.out.println("ISBN: " + myItem.isbn);
		}
	}
}
```

```
public class Book{

	public String title;
	public String author;
	public String isbn;
}
```

```
C:\Windows\system32\cmd.exe

---->Books:
Name: The Hobbit
Author: Tolkien, J. R. R.
ISBN: 0-261-10221-4

---->Books:
Name: Harry Potter and the Prisoner of Azkaban
Author: Rowling, J. K.
ISBN: 978-1594130021

---->Books:
Name: King, S.
Author: Misery
ISBN: 978-1444741292
Press any key to continue . . .
```

Activity 10.2

Amend the guitar program from the previous step, so that the code includes an array list of type `Guitar`. The program should add all of the objects to the array list and then use a FOR EACH loop to print all properties of each object to the command-line.

Dynamic Array Lists and Objects

By combining techniques learnt from previous chapters, it is possible to write some pretty 'nifty' code – code that is both dynamic and well designed. The program below allows a user to add a book object to an array list, and then print the entire content of the array list to the command-line. This time, however, the program features two methods: [addBook()] and [showBooks()]. The [addBook()] method requires three arguments to execute: they are author, title and ISBN. The method then creates a new book object and sets the properties of the book object to the passed arguments, before adding the object to the array list. Note that because the variable `x` is 'local', the variable is temporary and thus will only exist for the duration of the method. The [showBooks()] method features a FOR EACH loop which is used to print the properties of each object stored in the array list to the command-line.

Try the code

```
import java.io.*;
import java.util.ArrayList;
public class TestRun{
	private ArrayList<Book> myBooks = new ArrayList<Book>(); //array list
	public static void main(String[] args){
		BufferedReader y = new BufferedReader(new InputStreamReader(System.in));
		String myTitle = null; //variables to store user input
		String myAuthor = null;
		String myISBN = null;
		TestRun library = new TestRun(); //creates an object from the class
		while(true){ //program enters an infinite loop
			try{ //try to gather user input and catch any IO errors
				System.out.println("\n--->Please enter book title;");
				myTitle = y.readLine();
				System.out.println("\n--->Please enter book author;");
				myAuthor = y.readLine();
				System.out.println("\n--->Please enter book ISBN;");
				myISBN = y.readLine();
			}catch(IOException e){
				System.out.println("Error: " + e);
			}
			library.addBook(myTitle, myAuthor, myISBN); //call addBook method
			library.showBooks(); //call showBooks method

		}
	}

	public void addBook(String a_title, String a_author, String a_isbn){
		Book x = new Book(); //creates new book object
		x.title = a_title; //sets the title property to the passed argument
		x.author = a_author; //sets the author property to the passed argument
		x.isbn = a_isbn; //sets the isbn property to the passed argument
		myBooks.add(x); //adds the book object to the array list
	}

	public void showBooks(){
		for(Book myItem : myBooks){//cycles through the objects in the array list
		    System.out.println("\n---->Books:");
		    System.out.println("Title: " + myItem.title);//prints object title
		    System.out.println("Author: " + myItem.author);//prints object author
		    System.out.println("ISBN: " + myItem.isbn);//prints object isbn
		}
	}
}
```

```
C:\Windows\system32\cmd.exe

--->Please enter book name;
Best Guitars 2

--->Please enter book author;
Jason Bee

--->Please enter book ISBN;
234-78-57834

---->Books:
Name: Best Guitars 2
Author: Jason Bee
ISBN: 234-78-57834

--->Please enter book name;
```

Activity 10.3

Amend the guitar program from the previous step so that the program features two useful methods: [showGuitar()] and [addGuitar()]. The methods should be used to enable a user to add a guitar object to an array list, before printing the properties of each object in the list to the command-line.

Constructors and Classes

The world of object orientation is beautiful, and full of weird and wonderful little tricks. In the earlier part of this chapter, it was established how classes can be used to write custom data types which can then be used to closely describe objects found in the real world. There is another useful trick worth noting when defining custom data types, and that is the use of constructors. Constructors are used to request arguments before an object can be instantiated from a class; for example, a programmer can insist on a title and author argument before a book object can be created.

The syntax for using a constructor is demonstrated in the example below. Note that a constructor is given the same name as the class. The arguments that are required by the constructor (when an object is instantiated from the class) are written between the brackets, the same as writing parameters for a method (see Chapter 7). The [`this`] keyword is used to refer to a property or method that is accessible inside of an object; in this case, the [`this`] keyword is used to set the current object's title, author and ISBN properties to the values given as arguments to the constructor.

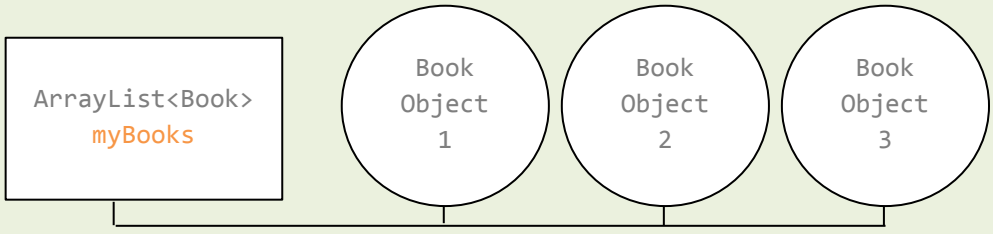
```
public class Book{

        public String title;
        public String author;
        public String isbn;

        public Book(String n, String a, String i){ //constructor
              this.title = n; //set an objects property to the given argument
              this.author = a;
              this.isbn = i;
        }
}
```

Once a constructor is defined, each time a new object is instantiated from the class the specified arguments must be given, else the compiler will throw an error. The syntax below illustrates a method that is used to create a new `Book` object, but this time with default values for the title, author and ISBN properties; the object is then added to an array list of type `Book`.

```
Private ArrayList<Book> myBooks = new ArrayList<Book>();

public void addBook(String a_title, String a_author, String a_isbn){
              myBooks.add(new Book(a_title, a_author, a_isbn));
}
```

Programming Tips!

Array lists are dynamic, and extremely useful, array types that can be used to store multiple objects:



The Pizza Program – Add, Edit, Delete and View

Congratulations, this is it, the 'final leg' of learning Java – it is now time to collect all of the techniques, knowledge and experience learnt in the previous chapters, and apply them to a given scenario – in particular, the pizza problem below! The given requirements below are purposely vague, so feel free to add any additional functionality (i.e. delivery charges, custom toppings, customer details and so on) that demonstrates skills learnt previously. Finally, let the creative energy flow and go wild!

Activity 10.4a

Pizza Program – write a program that enables a user to order pizzas from a set menu (the menu below). The program should also allow the user to edit, delete and view pizzas ordered. The program should utilise a custom pizza data type (a pizza class).

Activity 10.4b

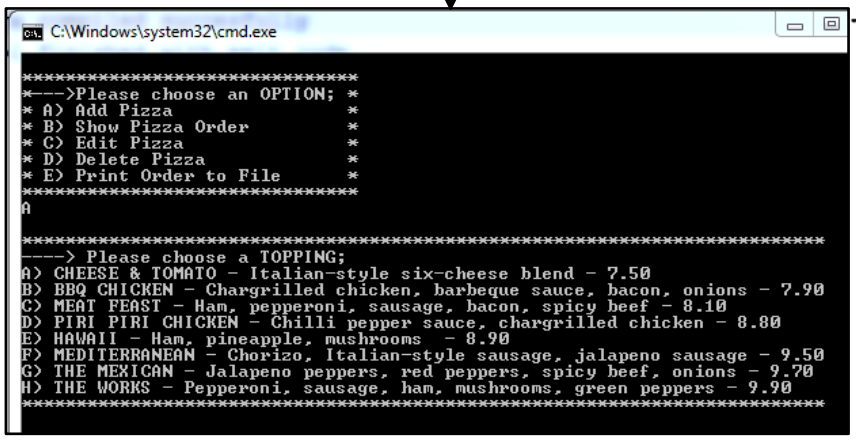
Pizza Program Extended – amend the program so that it implements appropriate validation (i.e. must choose a pizza from A to H, else the program repeats the question). Also, remove the possibility of the program encountering a run-time error by applying the [`try/catch`] technique as appropriate.

Activity 10.4c

Pizza Program Extended – amend the program so that the completed order can be written to a text file, stored in the same directory as the executable.

Menu

A. **CHEESE & TOMATO** – Italian-style six-cheese blend – £7.50
B. **BBQ CHICKEN** – Chargrilled chicken, barbeque sauce, bacon, onions – £7.90
C. **MEAT FEAST** – Ham, pepperoni, sausage, bacon, spicy beef – £8.10
D. **PIRI PIRI CHICKEN** – Chilli pepper sauce, chargrilled chicken – £8.80
E. **HAWAII** – Ham, pineapple, mushrooms – £8.90
F. **MEDITERRANEAN** – Chorizo, Italian–style sausage, jalapeno sausage – £9.50
G. **THE MEXICAN** – Jalapeno peppers, red peppers, spicy beef, onions – £9.70
H. **THE WORKS** – Pepperoni, sausage, ham, mushrooms, green peppers – £9.90



Programming Tips!

In earlier chapters the [`try/catch`] technique was used to catch unexpected errors when working with inputs and outputs; this is why the `IOException` error type was used when working with the [`readLine()`] method, i.e. [`catch(IOException ex)`]. However, there are many different errors types and thus a programmer should use the most appropriate type for the likely encountered error. That being said, at the top of the hierarchy is Exception which, despite being is a vague type, can be used to catch any run-time errors!

