Glasgow Caledonian University

School of Engineering and Computing

INTRODUCTION TO PROGRAMMING 2

M1G413283

07/01/11 JP

1. Designing a program

Introduction

In your course you have learned about the process of analysis and design of Computer Systems, and you have seen examples of the ways in which a system can be modelled using, for example, class diagrams. You have also seen how a **model** of the data in the system can be used to help design a database to store information which needs to be kept permanently.

A model can also be used to help design and create the software application, or **program**, which actually carries out the functions which the system is required to perform.

You have also learned some of the basic concepts and techniques for writing programs in the Java programming language. In this module you will see how the programming techniques you have learned can be used to create a working computer program. You will also learn some useful new programming techniques.

Java Programs

A Java program can be anything from a few lines of code which perform a single function, for example to add two numbers, to a large enterprise system consisting of many Java classes which provide very complex services, for example managing the customers and accounts of a bank.

Java is an **object-oriented** programming language. When an object-oriented program is running, it creates objects which work together, or **collaborate**, to perform the required actions. These objects often represent **entities** in the system. For example, the GCU

Tours system you have seen previously will have objects representing customers, bookings, and so on.

You can compare an object-oriented (OO) program to a team of people working in a restaurant. In the restaurant, each person has a specific job to do, while there may be several people doing the same type of job, waiters for example. The success of the restaurant depends on everyone doing their job, but also, crucially, on the communication between them. The waiters must pass orders to the kitchen; the head chef must coordinate the cooks to get the parts of each dish ready at the same time; and the kitchen must tell the waiters when dishes are ready to serve. Without this communication, the diners will be kept waiting and will not get the food they ordered. Similarly, objects in a computer system need to communicate, and the links between these objects are a key part of the system design.

An OO program creates the objects it needs as it runs. To create an object, it needs to have a **template** which specifies what **type of object** will be created and what that type of object can do. This template is called a **class**. In fact, when you write a program, you are actually writing the templates which are used to create objects. An object is a single instance of a class – there can be many objects of the same type.

A class specifies the following for the objects it is used to create:

- The **name** of the type of object (e.g. Customer)
- The **properties** which each object will have (e.g. name)
- The **actions** which each object can perform (e.g. change password)
- The way in which each object is **linked with other objects (**e.g. with Booking objects)

NOTE

Java is one of the most popular programming languages and is widely used in industry. Other popular object-oriented languages include **C#**, **Visual Basic.NET** and **C++**.

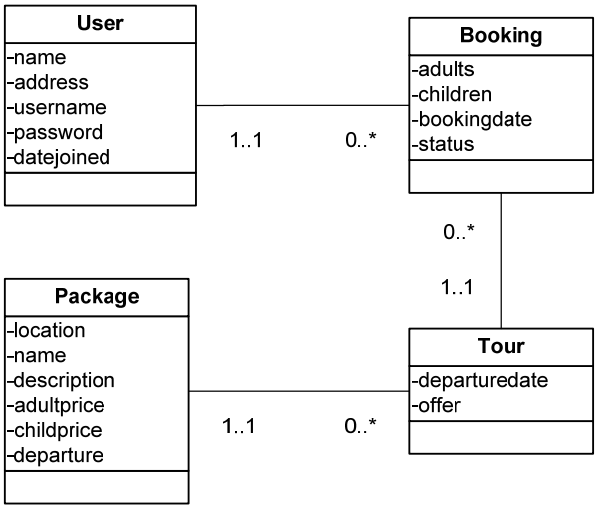
Although OO languages are now the most widely used, there are a number of other important types of language, including **procedural** languages like **C** and **functional** languages such as **Scheme**. Some languages are used for specific purposes, for example **JavaScript** and **PHP**, which are mainly used for web applications. There are also general purpose languages which have elements of OO and other language types, for example **Python**.

In your university career you will come across a number of languages. Many of the techniques which you learn with Java have equivalents in other languages, and once you learn Java it is relatively easy to adapt to others.

Classes

The following UML class diagram shows some classes in the GCU Tours system. These classes are typical of the entities you might find in a computer system used by a company to manage its business, and represent the data, or information, which is important to this particular company – users, bookings, and so on.

These particular classes will be implemented as Java classes which form part of a program which performs actions like creating bookings and registering users. The classes will also be implemented as database tables to store the information permanently.



Not all classes in a real system represent information, though. A working program needs objects which do other jobs, such as:

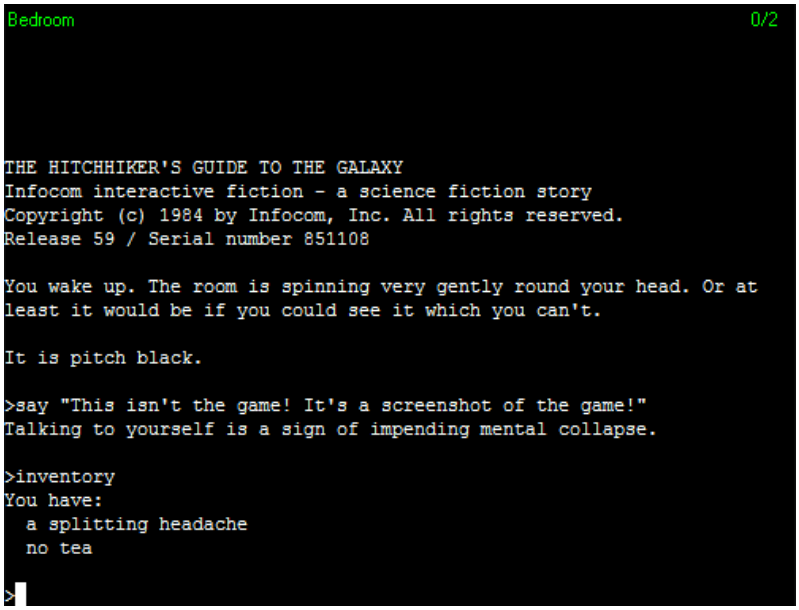
- Getting **input** from the user and displaying **output**
- **Controlling** the flow of activity while the program runs
- Other specialised tasks which help the program in some way, for example by formatting output, getting information into and out of the database, and so on

The GCU Tours system becomes quite complicated when all the classes are included. Instead, we will look at a simple example of a different kind of system, a game in fact, which uses objects which do a range of jobs and work together as a complete program.

The GCU adventure game

The GCU game is a text adventure game (sometimes known as interactive fiction). Text adventure games are a legacy from a time when computing power was small, when

terminal access was commonplace, and when monochrome graphics was "state of the art". Players used imagination, fired by descriptions of old abandoned caves, dungeons, and even futuristic spaceships populated by Vogons. For example, the screenshot shows the opening screen of the Hitchhiker's Guide to the Galaxy text adventure.



OK, our own game is really not very adventurous. In fact it's based on the "World of Zuul" game in Barnes and Kölling's book *Objects First* (highly recommended as additional reading), which the authors describe as the "world's most boring adventure game". It will do fine for our purposes, though.

The **game** centres around characters travelling through a world which consists of **rooms**. Each room has a **description**. Rooms have **exits** which lead to other rooms. Each room can contain several **items** which may be used by a character when in that room. An item will also have a description. A **player** navigates through a world by typing **commands** (*go north*, *go south*, *quit*, and so on). The game can be played by several players.

Objects in the adventure game

We can use the description above to make a first attempt at designing a program which will implement the game. What kind of objects will we need? The words highlighted in bold may help. These are all nouns, and name some 'things' which may be objects in the game. Let's look at these:

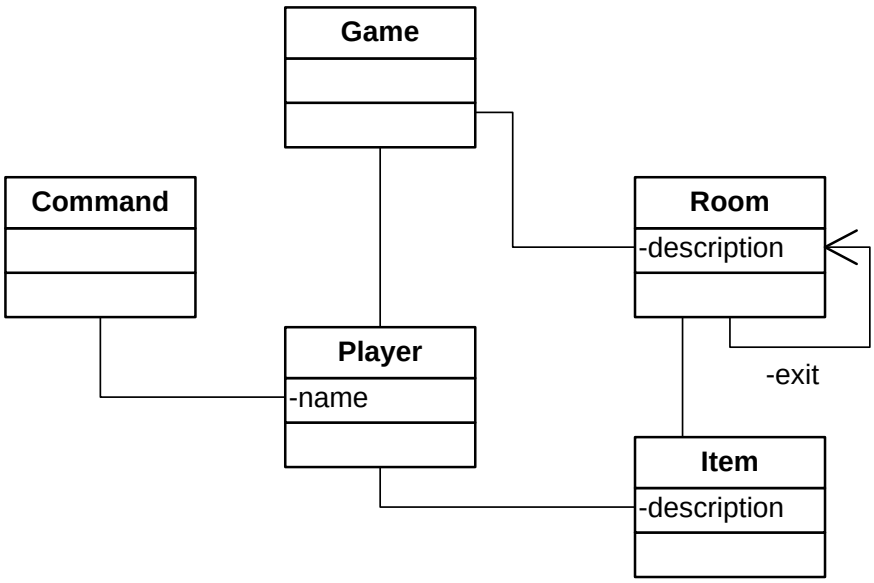
- **game** – this will be an object which sets up the game world and controls the game play. There will be **only one Game obj**ect.

- **room** – this will be an object which represents a room in the game world. There may be **many rooms**.

- **item** – this will be an object which represents an item in a room. There may be **several items** in each room.
- ***description*** – this simply describes a room or item, and will be a **property** rather than an object in its own right
- ***exit*** – an exit from a room is really just a way into another room, so an exit is actually a way of **referring to a room object**, not another type of object
- **player** – this will be an object which represents a player. A player will **be in one room** at any point in the game. There may be **several players** in the game. Each player should have a **name**, which is a property of the object.
- **command** – it may not be immediately obvious that a command will be an object, but the job of representing players' actions may be quite complicated. For example, we need to check whether the words the player types are in fact a valid command. Command objects will be useful, and there will be **one object for each input** entered by a player.

Modelling the adventure game

So let's look at a first attempt at a **model** for the adventure game. Here is a class diagram of the classes which will be needed to create these objects.



The diagram shows Game, Room, Item, Player and Command classes, with properties as described above. The **links**, or **associations**, between the classes show that **objects**

will interact in some way. For example, a game object will interact with player objects. We'll look at these interactions in more detail as we go along.

Note that the Room class is linked to itself – this is because we noted above that a room's exit is actually a reference to another room object, so rooms can be linked with each other

REMEMBER

A program creates **objects** to do its work – it can only create an object if a **class** has been written to provide a **template** for the object. If we want to have player objects we need to write a Player class to define what player objects are like.

The Player class will have a property called *name* – different player objects will have different values for the *name* property.

While it is not the only part of a model of a system, the class diagram is very important when you start to build the system as a Java program. The classes become the Java classes which you need to write. When we start creating our game in the next chapter, we will start by creating some classes in Java.

Other UML diagrams can be used to add details to the model. For example, **activity diagrams** are a kind of flow chart, and describe the flow of activity as the program runs. **Sequence diagrams** and **communication diagrams** model the details of interactions between objects. You have also seen **use case diagrams** which are used to model the requirements for the system. These models, and others, are related to the details of the code written within methods in the classes.

We will not be able to examine all these diagrams fully in this module.

Designing vs. implementing programs

You may be thinking that designing and modelling programs is difficult. You are right! It takes experience to do it well, and you will learn more about program design in future modules. In this module, we are mainly concerned with learning how to turn a model which has already been created into working code.

What's next?

In the lectures we will build incrementally a working implementation of the "World of GCU" adventure game. As we do so you will learn about some new programming techniques and tools as they are needed. You will also learn how the associations between classes in the game are implemented in Java.

For reference, at the start of each chapter of the lecture notes there will be a summary of the programming techniques and class associations which are introduced in that chapter, like this:

Programming techniques in this chapter:
....

Class associations in this chapter:
....

We will start by creating the Game and Player classes and testing the interaction between instances of these classes.