Start working on your first big Java project - a text-based virtual home tour.

**Background**

When we think of computer programs and applications today, we think of graphical interfaces with buttons and expandable menus and other kinds of displays. But the consumer computers started finding their ways into homes 5 years before the technology for what we would think of as "graphics" debuted, and it would take another 5 years for those first graphics cards to become commonplace. So for a solid decade, almost all software had to display output in a text format, and collect input from users through text-based commands.

Some of the first games for these home computers then were similar to Choose-Your-Own-Adventure books, and defined the "adventure" genre accordingly (sometimes the genre name is attributed to the game Colossal Cave Adventure, a pioneer of the format) . In these kinds of games, the player is presented with a description of a place or an event, and has to type in a command to do something. The game parses this command, and then prints some new output accordingly. Generally, the player is navigating through different "rooms" or areas using cardinal directions to explore the environment.

For a good example of this, check out the adventure game Zork, which you can play from the link in the reference section below. You'll find that while the complete list of commands was only printed in the manual, many of them are intuitive - **open mailbox, take leaflet, read leaflet, go south, look...**

For this project, you will be building a similar kind of application, but rather than an adventure game we'll just use it to present an interactive "tour" of a home - maybe your current home, maybe your dream home. In any case, the pattern will be exactly the same as other text-based games:

1. Display prompt
2. Collect input
3. Process input as a command
4. Print output of command

Here's an example of what your game might look like:

**Instructions**

For now, we're just going to create the shell for our project, based on what we understand of the requirements. Later, as your skills improve and you continue to learn, you'll probably make changes to these classes or their structure - a process called ***refactoring*** - to make your prject easier to expand.

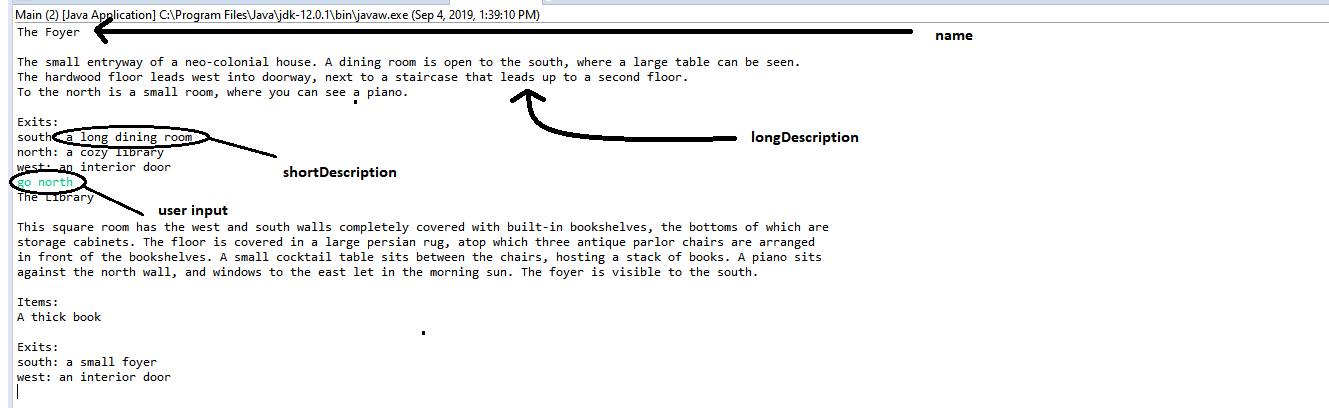
1. In Eclipse, create a project named HomeTour
2. In the HomeTour project, create the following packages and classes:
   1. fixtures
      1. fixtures.Fixture (abstract)
      2. fixtures.Room
   2. game
      1. game.Main
      2. game.Player
      3. game.RoomManager

These classes will work as follows:

**fixtures.Fixture:**  
This abstract class will be used as a base for anything that can be *looked* at or interacted with. This class should define (at least) the following properties:

* String name : a short name / title for the fixture
* String shortDescription : a one-sentence-long description of a fixture, used to briefly mention the fixture
* String longDescription : a paragraph-long description of the thing, displayed when the player investigates the fixture thoroughly (looks at it, or enters a room)

Here's an example of what your game might look like, so these make sense:



**fixtures.Room:**  
This class represents a room in the house. It will extend fixtures.Fixture, and so will inherit the descriptive properties. The Room will also have the following properties:

* Room[] exits : the rooms adjacent to this one. You might decide that a room in a particular direction always uses a certain index, e.g. a north exit always goes in index 0, an east exit always goes in index 1, etc. If so, then the size of this array depends on how many directions you want to support.

The Room class should also have a constructor that accepts a name, shortDescription, and longDescription. You might also find it convenient to create a getter not just for all the exits, but for a particular exit given a direction:

public Room(String name, String shortDescription, String longDescription) {

super(name, shortDescription, longDescription);

this.exits = new Room[?]; // size is your choice

}

public Room[] getExits() {

}

public Room getExit(String direction) {

}

**game.Main:**  
This class will store the main(String[]) method for our game (and of course, it will be the only class that has a main(String[]) method). This is where the game-loop will go, where we'll display a prompt, collect input, and parse that input

The printRoom(Player) method will print a prompt to the console for the player's current room, similar to the above image.

The collectInput() method will use a Scanner object to collect console input from the user, and then will divide that input into multiple parts. *Generally* those parts will look like this:

1. An action
2. The target of an action (if any)

For example, "go east" -> "go" is the command, "east" is the target. This method will break the input into a String[], and return that.

The parse(String[], Player) method will take the output of the above collectInput() method and a player object, and will resolve that command. This can actually be simpler than it sounds - the first index of the passed-in String[] should be the *action*, so you can switch on that and handle the target differently for each case. The Player object is there so you can modify it if needed (like changing the Player's currentRoom based on the direction moved)

public static void main(String[] args) {

}

private static void printRoom(Player player) {

}

private static String[] collectInput() {

}

private static void parse(String[] command, Player player) {

}

**game.Player:**  
This class represents the player moving through these rooms. The Player class has these properties:

* Room currentRoom : the room the player is currently in.

**game.RoomManager:**  
This class will be responsible for "loading" our rooms into memory. When game.Main is executed, it will invoke the init() method in this class that will instantiate all our Room objects, link them together as exits, and designate a startingRoom.

* Room startingRoom : the room a player should start in.
* Room[] rooms : all the rooms in the house.

public void init() {

Room foyer = new Room(

"The Foyer",

"a small foyer",

"The small entryway of a neo-colonial house. A dining room is open to the south, where a large table can be seen." + "\n"

+ "The hardwood floor leads west into doorway, next to a staircase that leads up to a second floor." + "\n"

+ "To the north is a small room, where you can see a piano.");

this.rooms[0] = foyer;

this.startingRoom = foyer;

}

That's it for now! As you continue to learn more about the Java language, you'll fill in more of this application.

REF VIDEO

Play Zork

https://classicreload.com/zork-i.html