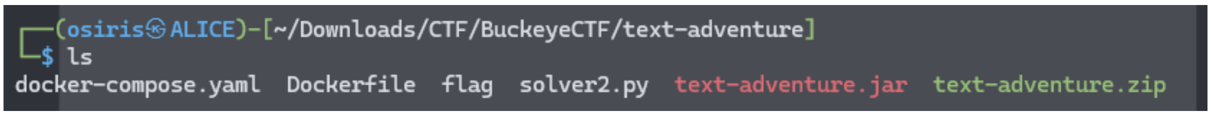
# REV

## TEXT-ADVENTURE

**Description:** I just wrote a text adventure game after learning Java, but maybe I should've added some instructions....

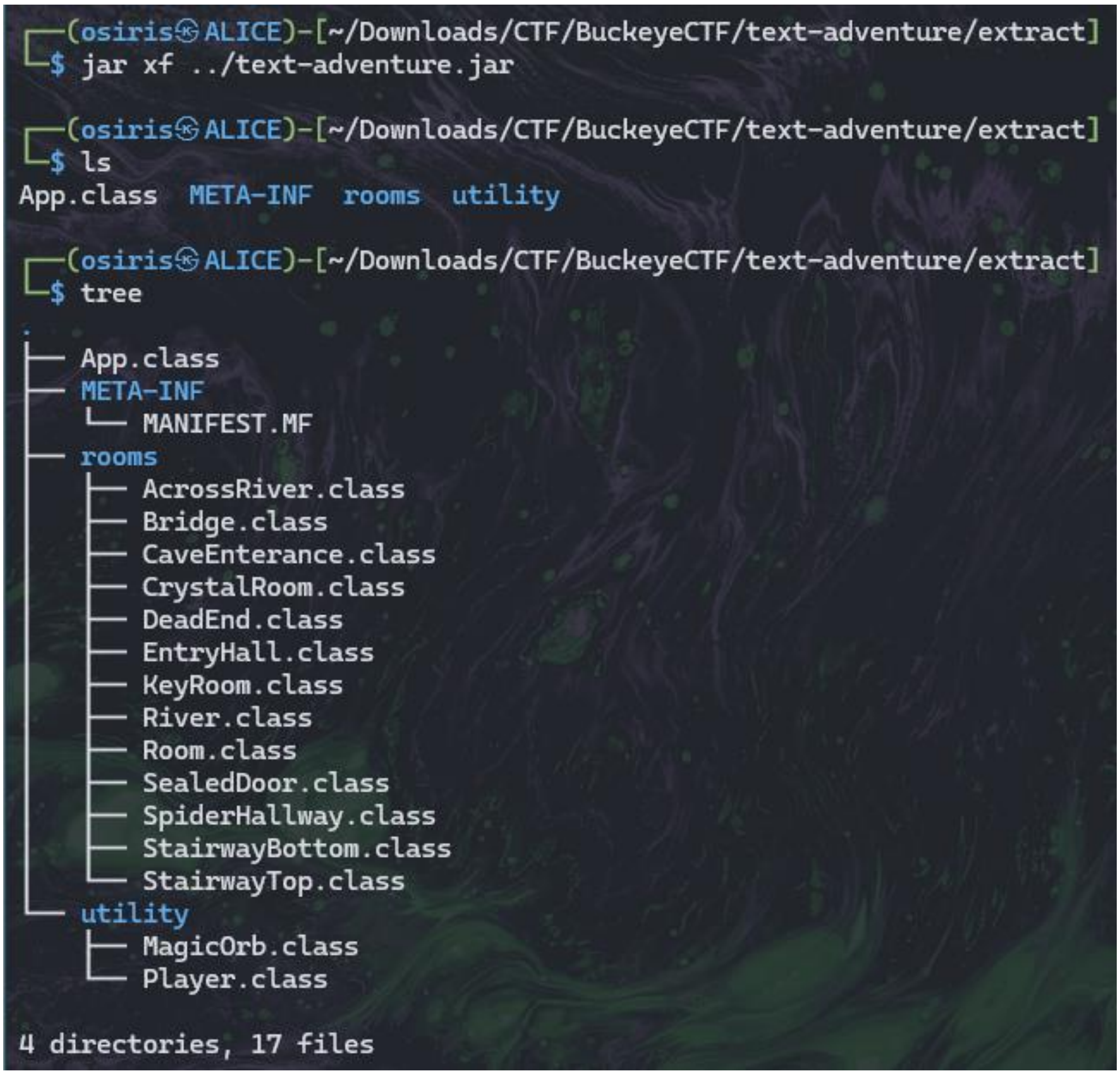
**Given File:**

We are provided with a .jar file containing a text adventure game. The task involves extracting and analyzing the code to find the flag hidden inside.



```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure]
└─$ ls
docker-compose.yaml  Dockerfile  flag  solver2.py  text-adventure.jar  text-adventure.zip
```

First, we extract the contents of the **.jar** file to view the internal structure. The structure contains several .class files:

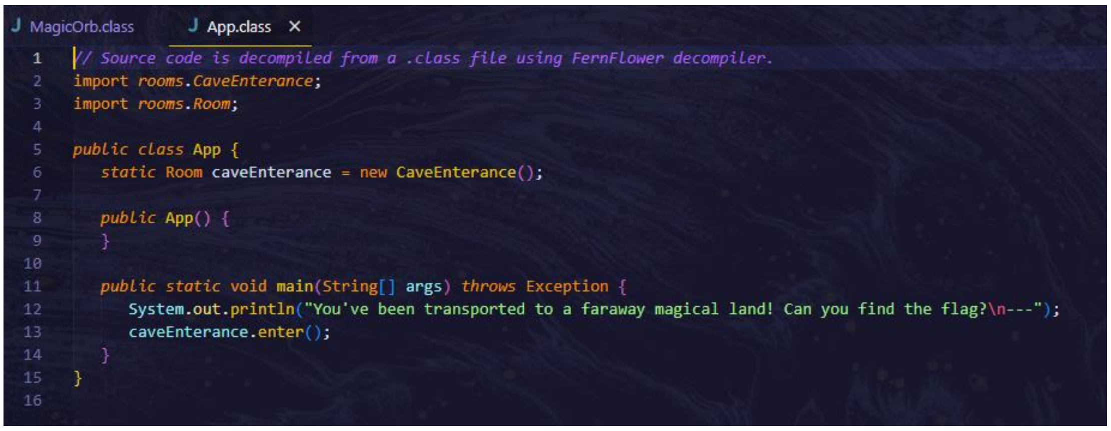


```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract]
└─$ jar xf ../text-adventure.jar

┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract]
└─$ ls
App.class  META-INF  rooms  utility

┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract]
└─$ tree
.
├── App.class
├── META-INF
│   └── MANIFEST.MF
├── rooms
│   ├── AcrossRiver.class
│   ├── Bridge.class
│   ├── CaveEnterance.class
│   ├── CrystalRoom.class
│   ├── DeadEnd.class
│   ├── EntryHall.class
│   ├── KeyRoom.class
│   ├── River.class
│   ├── Room.class
│   ├── SealedDoor.class
│   ├── SpiderHallway.class
│   ├── StairwayBottom.class
│   └── StairwayTop.class
└── utility
    ├── MagicOrb.class
    └── Player.class

4 directories, 17 files
```

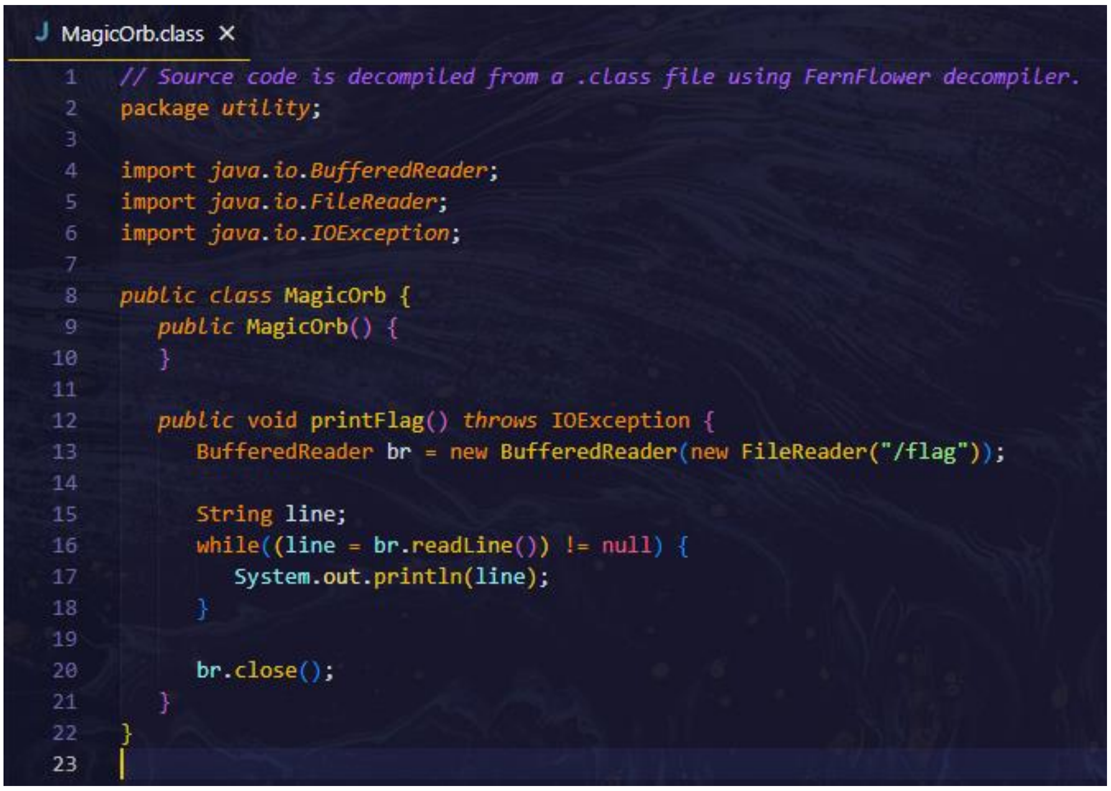
Using grep, we search through all files for occurrences of the word "**flag**". This helps identify which classes might be involved in revealing the flag. The output shows that the **MagicOrb.class, DeadEnd.class,** and **App.class** files reference the flag.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract]
└─$ grep -rn "flag" .
grep: ./rooms/DeadEnd.class: binary file matches
grep: ./App.class: binary file matches
grep: ./utility/MagicOrb.class: binary file matches
```

- The entry point (**void main**) is located in **App.class**.
- The flag is handled within **MagicOrb.class**, likely making it the final stop of the program due to its **close()** function.

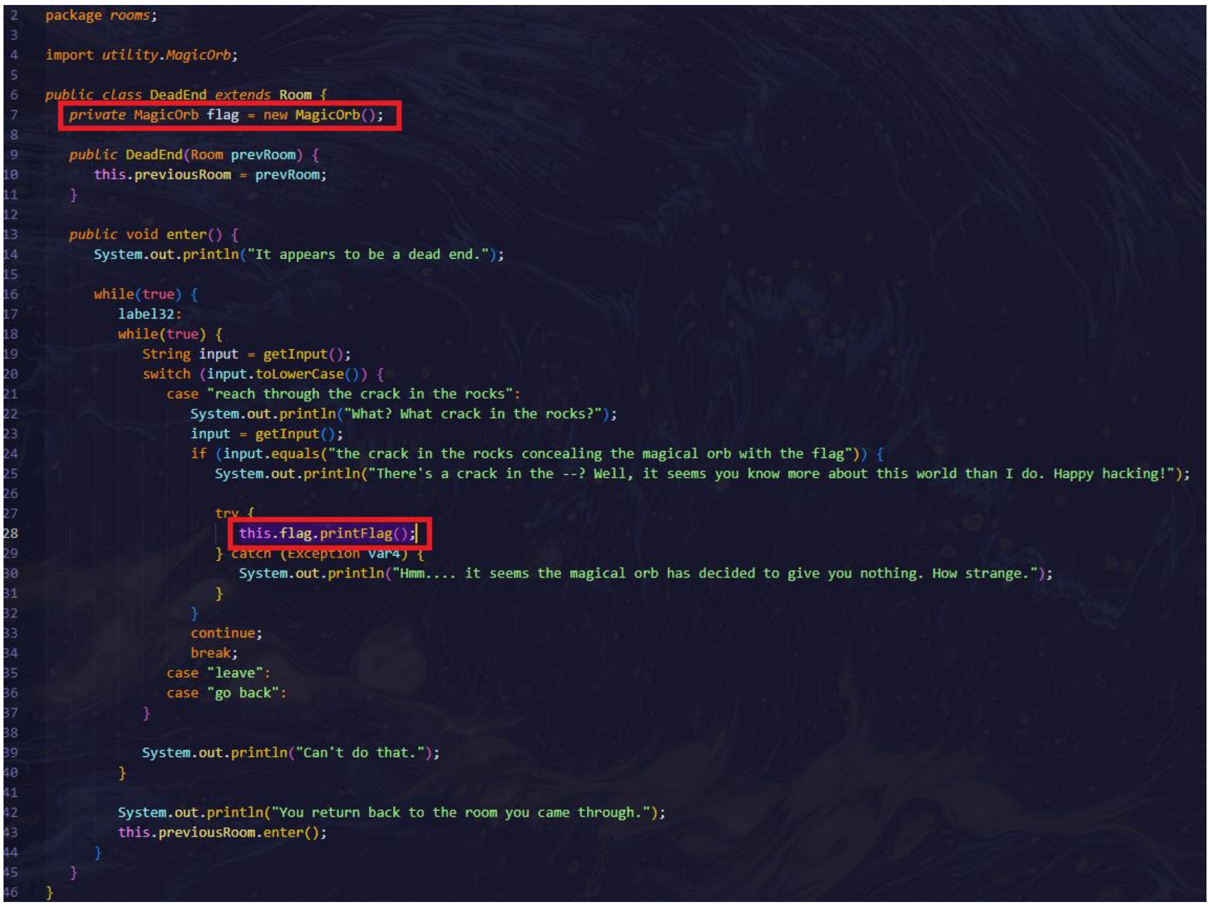


```java
// Source code is decompiled from a .class file using FernFlower decompiler.
import rooms.CaveEnterance;
import rooms.Room;

public class App {
   static Room caveEnterance = new CaveEnterance();

   public App() {
   }

   public static void main(String[] args) throws Exception {
      System.out.println("You've been transported to a faraway magical land! Can you find the flag?\n---");
      caveEnterance.enter();
   }
}
```



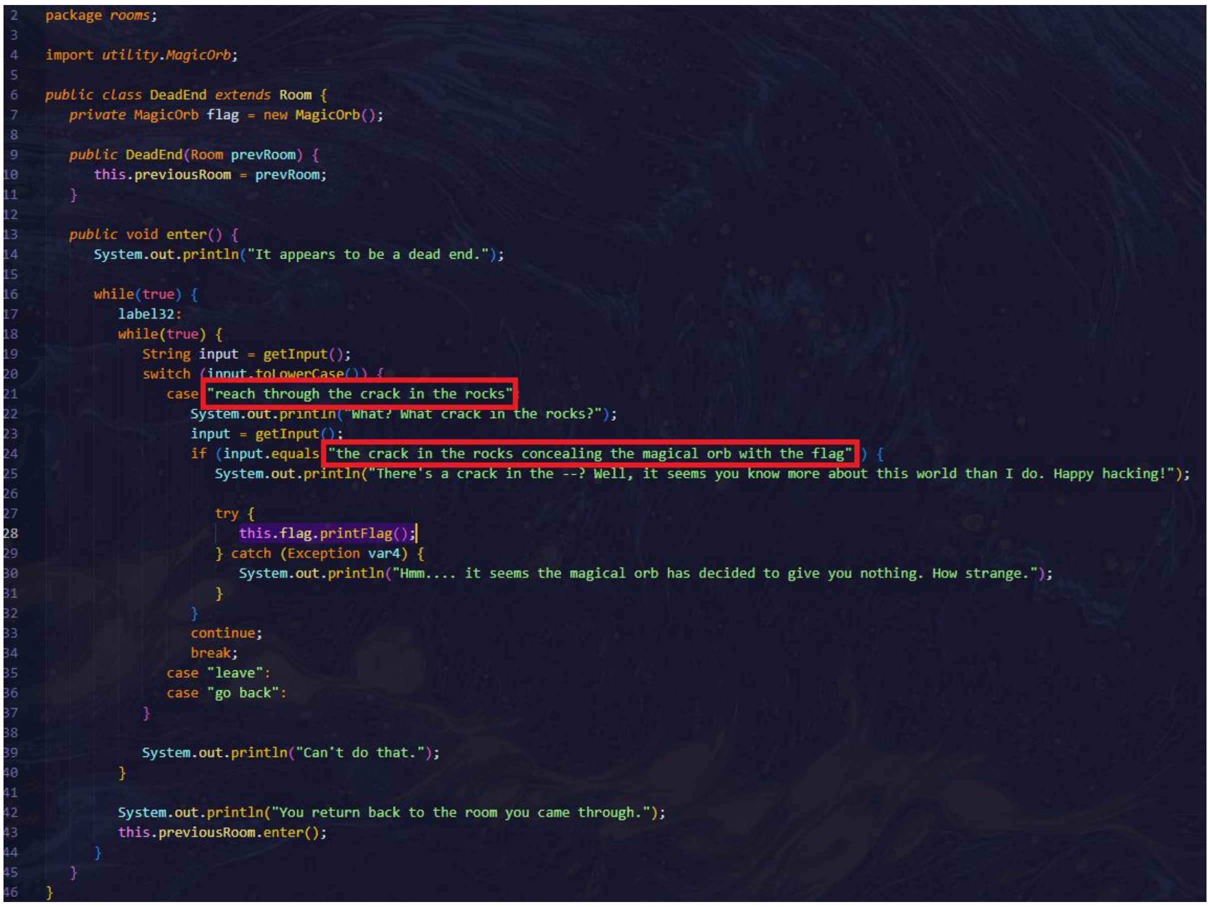
```java
// Source code is decompiled from a .class file using FernFlower decompiler.
package utility;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class MagicOrb {
   public MagicOrb() {
   }

   public void printFlag() throws IOException {
      BufferedReader br = new BufferedReader(new FileReader("/flag"));

      String line;
      while((line = br.readLine()) != null) {
         System.out.println(line);
      }

      br.close();
   }
}
```

We can start from the end of the program (where the flag is loaded) and reverse the flow to understand how the player reaches the flag. The **MagicOrb.class** likely holds the flag, and the **DeadEnd.class** seems to be the last class that triggers the program flow.



```java
package rooms;

import utility.MagicOrb;

public class DeadEnd extends Room {
   private MagicOrb flag = new MagicOrb();

   public DeadEnd(Room prevRoom) {
      this.previousRoom = prevRoom;
   }

   public void enter() {
      System.out.println("It appears to be a dead end.");

      while(true) {
         label32:
         while(true) {
            String input = getInput();
            switch (input.toLowerCase()) {
               case "reach through the crack in the rocks":
                  System.out.println("What? What crack in the rocks?");
                  input = getInput();
                  if (input.equals("the crack in the rocks concealing the magical orb with the flag")) {
                     System.out.println("There's a crack in the --? Well, it seems you know more about this world than I do. Happy hacking!");

                     try {
                        this.flag.printFlag();
                     } catch (Exception var4) {
                        System.out.println("Hmm.... it seems the magical orb has decided to give you nothing. How strange.");
                     }
                  }
                  continue;
                  break;
               case "leave":
               case "go back":
            }

            System.out.println("Can't do that.");
         }

         System.out.println("You return back to the room you came through.");
         this.previousRoom.enter();
      }
   }
}
```

MagicOrb=flag. Now can confirm.

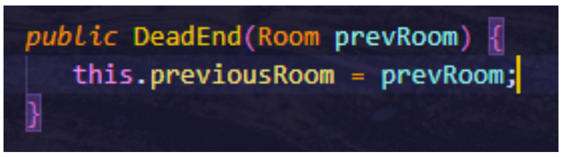


```java
package rooms;

import utility.MagicOrb;

public class DeadEnd extends Room {
   private MagicOrb flag = new MagicOrb();

   public DeadEnd(Room prevRoom) {
      this.previousRoom = prevRoom;
   }

   public void enter() {
      System.out.println("It appears to be a dead end.");

      while(true) {
         label32:
         while(true) {
            String input = getInput();
            switch (input.toLowerCase()) {
               case "reach through the crack in the rocks":
                  System.out.println("What? What crack in the rocks?");
                  input = getInput();
                  if (input.equals("the crack in the rocks concealing the magical orb with the flag")) {
                     System.out.println("There's a crack in the --? Well, it seems you know more about this world than I do. Happy hacking!");

                     try {
                        this.flag.printFlag();
                     } catch (Exception var4) {
                        System.out.println("Hmm.... it seems the magical orb has decided to give you nothing. How strange.");
                     }
                  }
                  continue;
                  break;
               case "leave":
               case "go back":
            }

            System.out.println("Can't do that.");
         }

         System.out.println("You return back to the room you came through.");
         this.previousRoom.enter();
      }
   }
}
```

To reverse the flow order, we extract the relevant flow of commands:

| |
|---|
| the crack in the rocks concealing the magical orb with the flag |
| reach through the crack in the rocks |

With the knowledge that **DeadEnd** represents the final stage of the program flow, the next step is to identify the class that loads the **DeadEnd**. This indicates that the preceding class represents the penultimate step in the program's sequence.

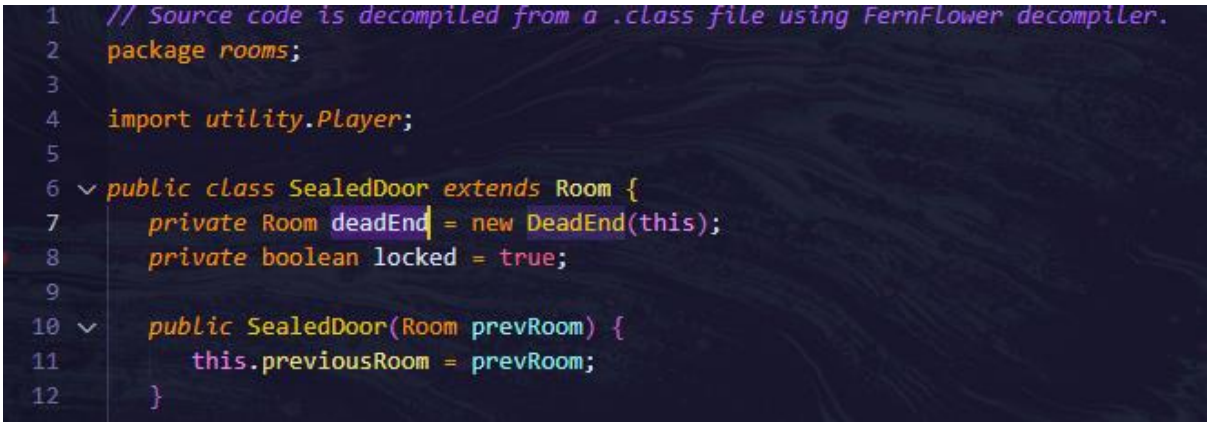


```java
public DeadEnd(Room prevRoom) {
   this.previousRoom = prevRoom;
}
```

To identify the previous room, we can use tools like Visual Studio's "find" function or the grep command. For instance, using grep to search for the keyword "DeadEnd" in the directory:

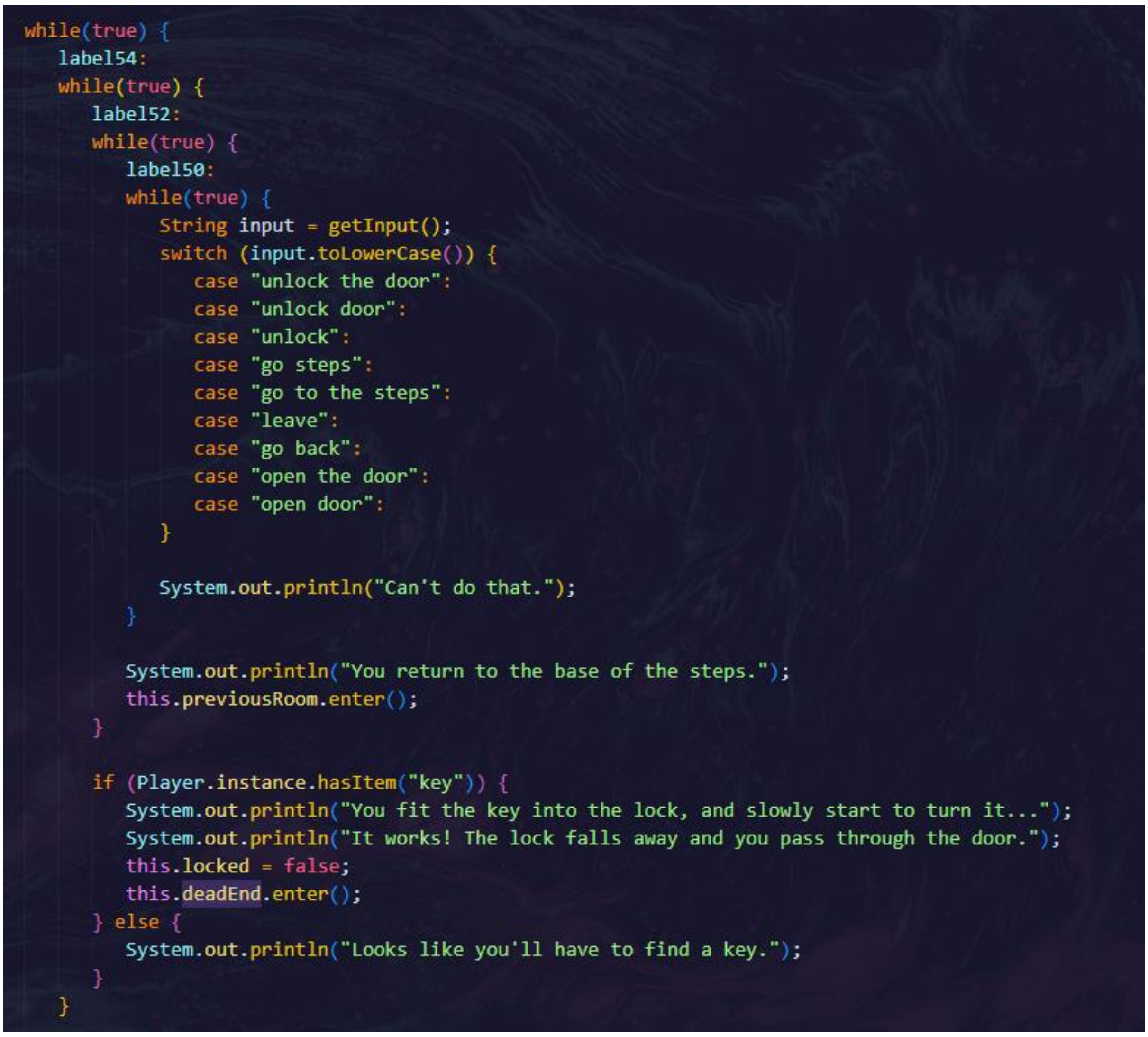
```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-
adventure/extract/rooms]
└─$ grep -r "DeadEnd"
grep: SealedDoor.class: binary file matches
grep: DeadEnd.class: binary file matches
```

The results show matches in both **SealedDoor.class** and **DeadEnd.class**. Since **SealedDoor** contains the reference to **DeadEnd**, this suggests that **SealedDoor** calls the **DeadEnd** class. Therefore, the program flow transitions from **SealedDoor** to **DeadEnd.** Next, you would load the **SealedDoor.class** to investigate its logic.



```
// Source code is decompiled from a .class file using FernFlower decompiler.
package rooms;

import utility.Player;

public class SealedDoor extends Room {
   private Room deadEnd = new DeadEnd(this);
   private boolean locked = true;

   public SealedDoor(Room prevRoom) {
      this.previousRoom = prevRoom;
   }
```

We can see that the DeadEnd class is referenced in the SealedDoor class. Next, we need to extract the possible inputs or commands that can be used within the SealedDoor class to interact with the DeadEnd.



```
while(true) {
   label54:
   while(true) {
      label52:
      while(true) {
         label50:
         while(true) {
            String input = getInput();
            switch (input.toLowerCase()) {
               case "unlock the door":
               case "unlock door":
               case "unlock":
               case "go steps":
               case "go to the steps":
               case "leave":
               case "go back":
               case "open the door":
               case "open door":
            }

            System.out.println("Can't do that.");
         }

         System.out.println("You return to the base of the steps.");
         this.previousRoom.enter();
      }

      if (Player.instance.hasItem("key")) {
         System.out.println("You fit the key into the lock, and slowly start to turn it...");
         System.out.println("It works! The lock falls away and you pass through the door.");
         this.locked = false;
         this.deadEnd.enter();
      } else {
         System.out.println("Looks like you'll have to find a key.");
      }
   }
```

The situation requires us to unlock the door, which is contingent on possessing a specific item, the key. To understand the flow of the game better, we will assume we have the key and reverse our way through the previous rooms.

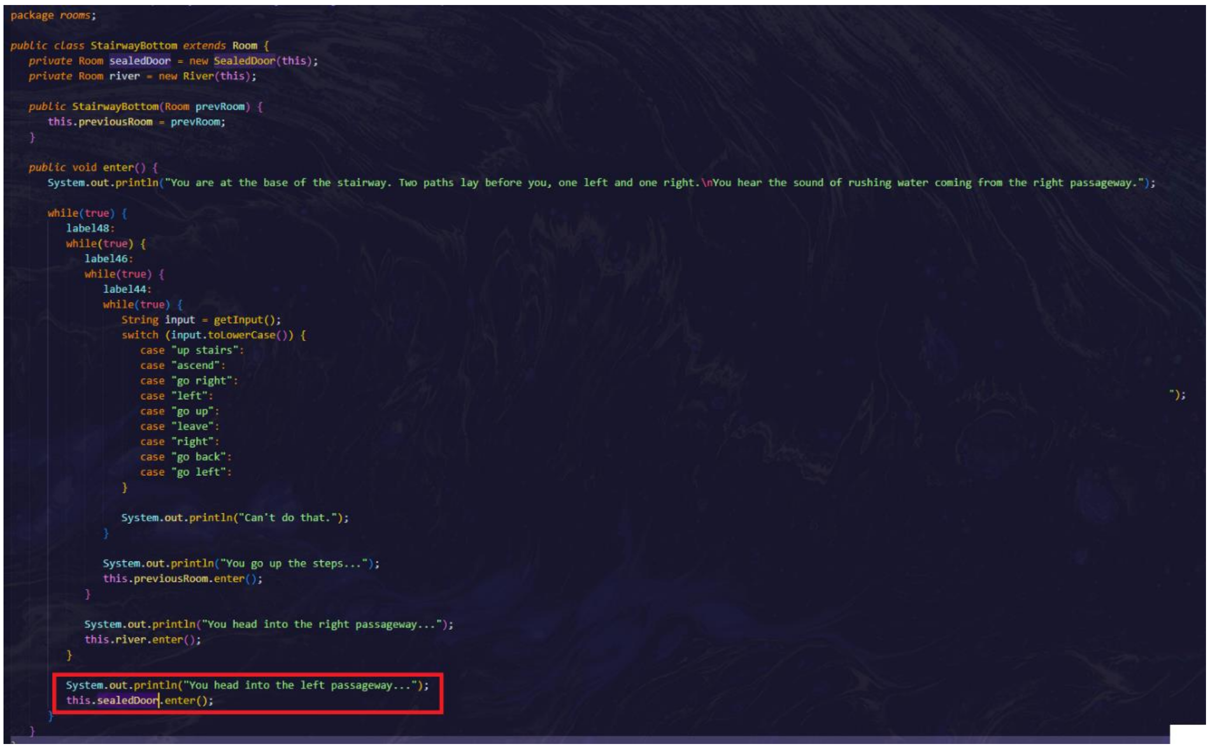
First, we need to find out what leads us into the SealedDoor room. To do this, we run a grep command to check where SealedDoor is referenced:

```
Unlock
```

Like usual. We need to find previous door to find how does it enter this room.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "SealedDoor"
grep: SealedDoor.class: binary file matches
grep: StairwayBottom.class: binary file matches
```

StairwayBottom will be the previous room.



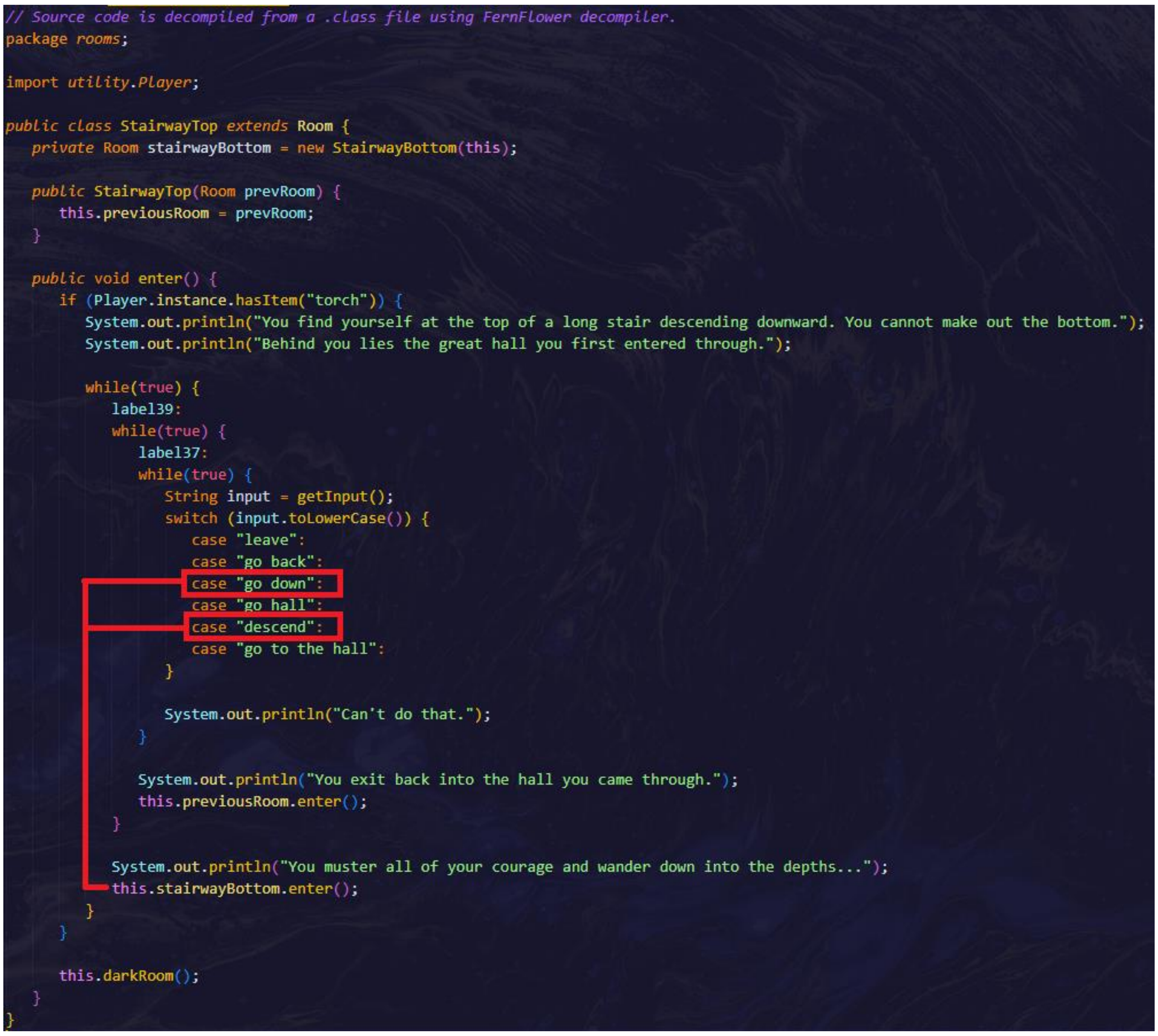
```
package rooms;

public class StairwayBottom extends Room {
   private Room sealedDoor = new SealedDoor(this);
   private Room river = new River(this);

   public StairwayBottom(Room prevRoom) {
      this.previousRoom = prevRoom;
   }

   public void enter() {
      System.out.println("You are at the base of the stairway. Two paths lay before you, one left and one right.\nYou hear the sound of rushing water coming from the right passageway.");

      while(true) {
         label48:
         while(true) {
            label46:
            while(true) {
               label44:
               while(true) {
                  String input = getInput();
                  switch (input.toLowerCase()) {
                     case "up stairs":
                     case "ascend":
                     case "go right":
                     case "left":                                                                              ");
                     case "go up":
                     case "leave":
                     case "right":
                     case "go back":
                     case "go left":
                  }

                  System.out.println("Can't do that.");
               }

               System.out.println("You go up the steps...");
               this.previousRoom.enter();
            }

            System.out.println("You head into the right passageway...");
            this.river.enter();
         }

         System.out.println("You head into the left passageway...");
         this.sealedDoor.enter();
      }
   }
```

SealedDoor will be load if we enter left. Let's save it in our list and proceed to previous room

```
go left
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "StairwayBottom"
grep: StairwayTop.class: binary file matches
grep: StairwayBottom.class: binary file matches
```

StairwayTop will be our previous room



```
// Source code is decompiled from a .class file using FernFlower decompiler.
package rooms;

import utility.Player;

public class StairwayTop extends Room {
   private Room stairwayBottom = new StairwayBottom(this);

   public StairwayTop(Room prevRoom) {
      this.previousRoom = prevRoom;
   }

   public void enter() {
      if (Player.instance.hasItem("torch")) {
         System.out.println("You find yourself at the top of a long stair descending downward. You cannot make out the bottom.");
         System.out.println("Behind you lies the great hall you first entered through.");

         while(true) {
            label39:
            while(true) {
               label37:
               while(true) {
                  String input = getInput();
                  switch (input.toLowerCase()) {
                     case "leave":
                     case "go back":
                     case "go down":
                     case "go hall":
                     case "descend":
                     case "go to the hall":
                  }

                  System.out.println("Can't do that.");
               }

               System.out.println("You exit back into the hall you came through.");
               this.previousRoom.enter();
            }

            System.out.println("You muster all of your courage and wander down into the depths...");
            this.stairwayBottom.enter();
         }
      }

      this.darkRoom();
   }
}
```

In order to proceed in the game, both the torch and the key are necessary items. The commands **"go down"** or **"descend"** should lead the player to StairwayBottom. These commands do not trigger any immediate actions (as the corresponding case statements are empty), but they allow the program to bypass the **"Can't do that."** message.

If the player types any of the valid commands listed (such as **"leave," "go back," "go hall,"** or **"go to the hall"**), the program will enter the second while(true) block, which represents the transition to the previousRoom. Notable extract:

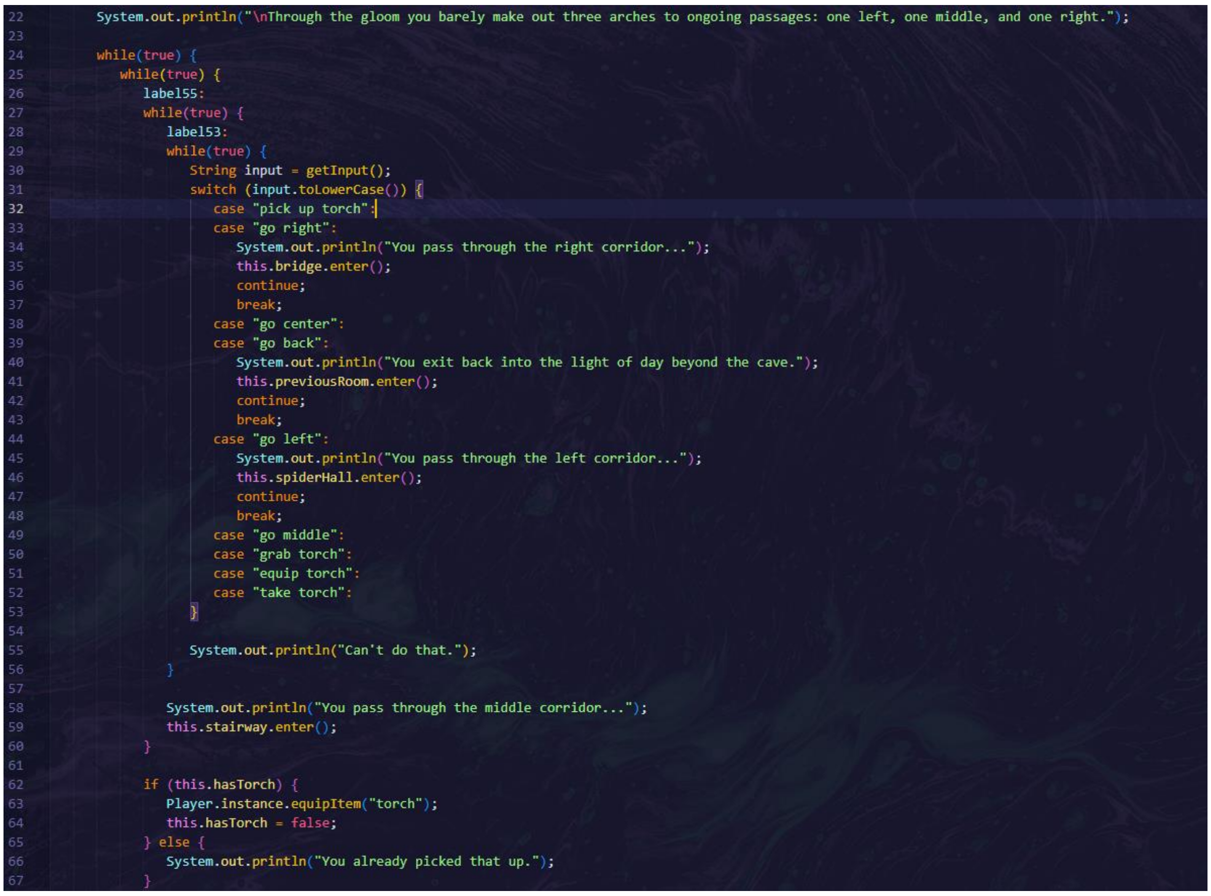
```
go down
```

Proceeding:

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "StairwayTop"
grep: StairwayTop.class: binary file matches
```

```
grep: EntryHall.class: binary file matches
```

Reading the EntryHall:



```java
      System.out.println("\nThrough the gloom you barely make out three arches to ongoing passages: one left, one middle, and one right.");

      while(true) {
         while(true) {
            label55:
            while(true) {
               label53:
               while(true) {
                  String input = getInput();
                  switch (input.toLowerCase()) {
                     case "pick up torch":
                     case "go right":
                        System.out.println("You pass through the right corridor...");
                        this.bridge.enter();
                        continue;
                        break;
                     case "go center":
                     case "go back":
                        System.out.println("You exit back into the light of day beyond the cave.");
                        this.previousRoom.enter();
                        continue;
                        break;
                     case "go left":
                        System.out.println("You pass through the left corridor...");
                        this.spiderHall.enter();
                        continue;
                        break;
                     case "go middle":
                     case "grab torch":
                     case "equip torch":
                     case "take torch":
                  }

                  System.out.println("Can't do that.");
               }

               System.out.println("You pass through the middle corridor...");
               this.stairway.enter();
            }

            if (this.hasTorch) {
               Player.instance.equipItem("torch");
               this.hasTorch = false;
            } else {
               System.out.println("You already picked that up.");
            }
```

This stage may present challenges due to the numerous commands available and the uncertainty regarding the correct path. However, it is clear that **"go middle"** represents our point of origin, and both the key and the torch are essential for progression. The other paths (right and left) may conceal the key, making it crucial to locate it. Notable command of reversing:

```
go middle
```

Now proceed find the key end point:

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-
adventure/extract/rooms]
└─$ grep -r "key"
grep: SealedDoor.class: binary file matches
grep: KeyRoom.class: binary file matches
grep: SpiderHallway.class: binary file matches
```

The **SealedDoor** will be the room where we need to unlock the key. Between the **KeyRoom** and **SpiderHallway**, we can employ similar commands such as **"take key"** or **"pickup key,"** among others.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-
adventure/extract/rooms]
└─$ grep -r "take key"
grep: KeyRoom.class: binary file matches
```

The **KeyRoom** is where the key is stored. We need to establish a clear order from this point onward. The current sequence of commands for reversing is as follows:

```
go middle
go down
go left
unlock
reach through the crack in the rocks
the crack in the rocks concealing the magical orb with the flag
```

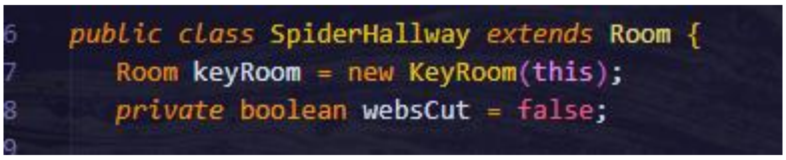
Let’s save that information for now. Next, we will outline the path to reverse the order for obtaining the key. We know that the **KeyRoom** is the endpoint for picking up the key. To determine the route to the **KeyRoom**, we can identify that the preceding room is **SpiderHallway**, as indicated by the previous grep command results.

Notable Command:

```
pick up key
```

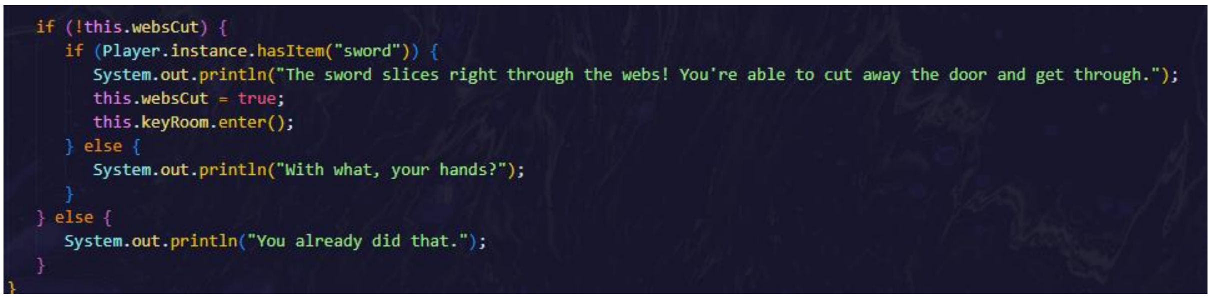
```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-
adventure/extract/rooms]
└─$ grep -r "KeyRoom"
grep: KeyRoom.class: binary file matches
grep: SpiderHallway.class: binary file matches
```

In spiderhallway. We need something to websCut turns true.



```java
public class SpiderHallway extends Room {
    Room keyRoom = new KeyRoom(this);
    private boolean websCut = false;
```

To set **websCut** to true, we need a sword to make the cut. Once we are able to cut, we can access the **KeyRoom**. Now, we need to determine how to obtain the sword. But first, let's examine the previous room.



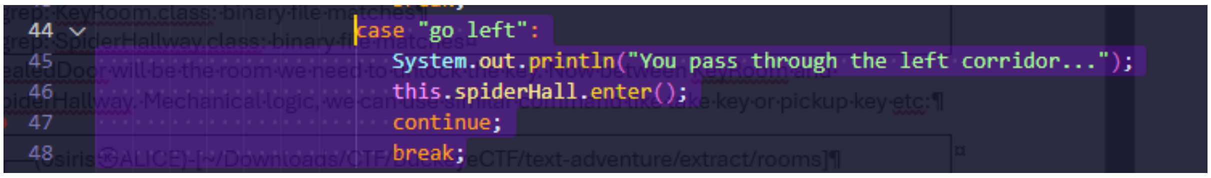
```java
  if (!this.websCut) {
    if (Player.instance.hasItem("sword")) {
      System.out.println("The sword slices right through the webs! You're able to cut away the door and get through.");
      this.websCut = true;
      this.keyRoom.enter();
    } else {
      System.out.println("With what, your hands?");
    }
  } else {
    System.out.println("You already did that.");
  }
}
```

Notable command:

```
cut
```

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "SpiderHallway"
grep: EntryHall.class: binary file matches
grep: SpiderHallway.class: binary file matches
```

The **EntryHall** will be our starting point. By checking the direction to **SpiderHall**, we see that the command **"go left"** will lead us into that room. However, we currently do not possess the sword needed to proceed further.



```
case "go left":
    System.out.println("You pass through the left corridor...");
    this.spiderHall.enter();
    continue;
    break;
```

Notable command:

```
go left
```

So collecting the previous order command (assume we have sword):

```
Go back
Go middle
Go down
go left
cut
pick up key
```

Save the second order command for now.

Finding sword:

Like usual. Use mechanism of pick up sword or take sword

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "take sword"
grep: AcrossRiver.class: binary file matches
```

So the endpoint of picking up the sword will be across river. We can just use that ‘take sword’ as our command that we already have sword set condition as true. Command:

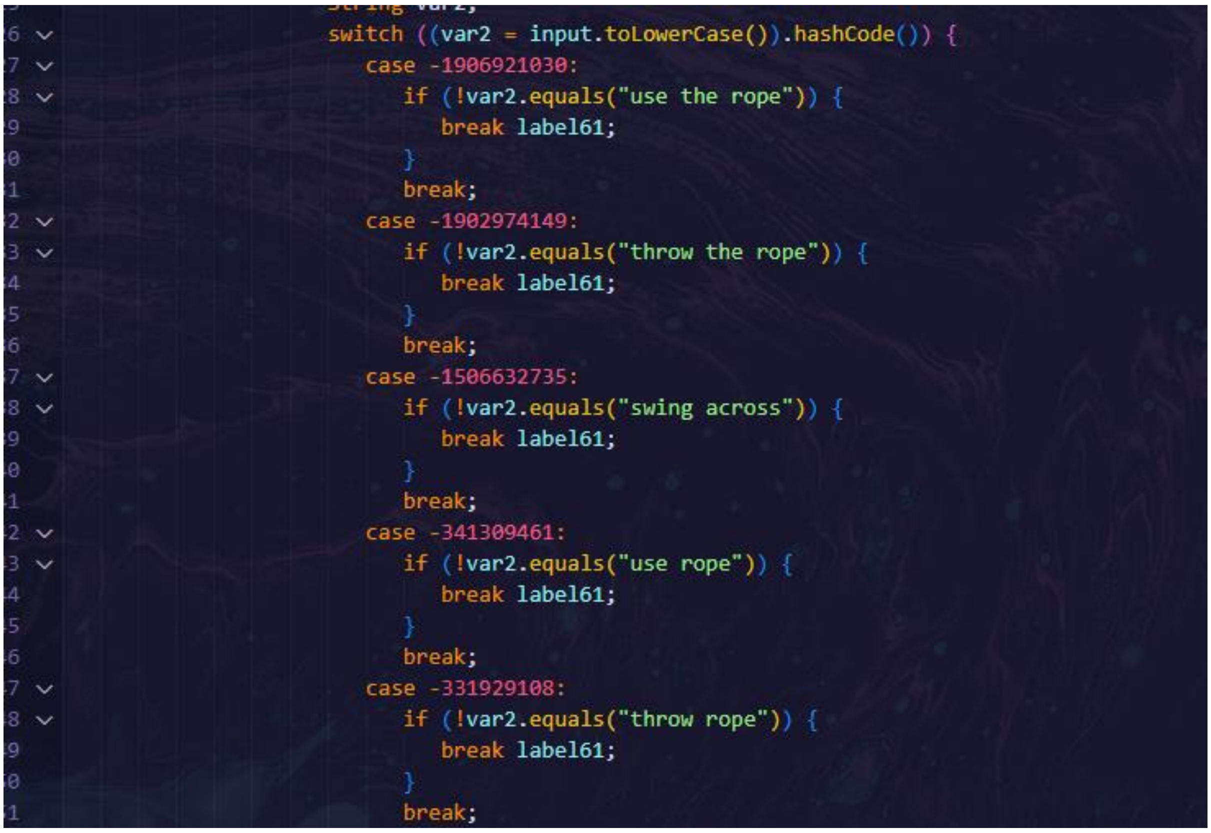
```
Take sword
```

Finding previous room of AcrossRiver.class

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "AcrossRiver"
```

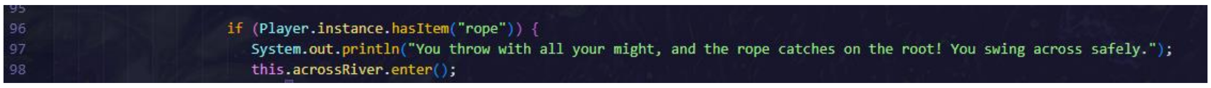
```
grep: AcrossRiver.class: binary file matches
grep: River.class: binary file matches
```

Reading river.class:



```
switch ((var2 = input.toLowerCase()).hashCode()) {
   case -1906921030:
      if (!var2.equals("use the rope")) {
         break label61;
      }
      break;
   case -1902974149:
      if (!var2.equals("throw the rope")) {
         break label61;
      }
      break;
   case -1506632735:
      if (!var2.equals("swing across")) {
         break label61;
      }
      break;
   case -341309461:
      if (!var2.equals("use rope")) {
         break label61;
      }
      break;
   case -331929108:
      if (!var2.equals("throw rope")) {
         break label61;
      }
      break;
```

Few command such as rope.



```
if (Player.instance.hasItem("rope")) {
   System.out.println("You throw with all your might, and the rope catches on the root! You swing across safely.");
   this.acrossRiver.enter();
```

To go acrossRiver we must have a rope. Now we need rope. Hmm.

Notable Command:

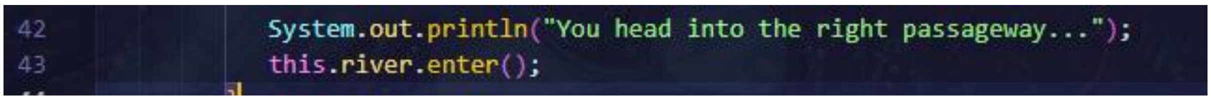
```
use rope
```

Finding the entry point(previous room):

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "River"
grep: AcrossRiver.class: binary file matches
grep: StairwayBottom.class: binary file matches
grep: River.class: binary file matches
```

We already went through acrossriver and river. Stairwaybottom will be our previous

**Stairwaybottom:**



```
System.out.println("You head into the right passageway...");
this.river.enter();
```

The command **"go right"** will serve as our entry point to the **River**. Since we have already entered the **StairwayBottom,** we are aware of the direction leading back to the **EntryHall**. Notable command: **"go right."**:

```
Go right
```

Collecting command to take the sword (assume we have rope):

```
Go back
Go middle
Go down
Go right
Use rope
Take sword
```

Finding rope:

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "take rope"
grep: CrystalRoom.class: binary file matches
```

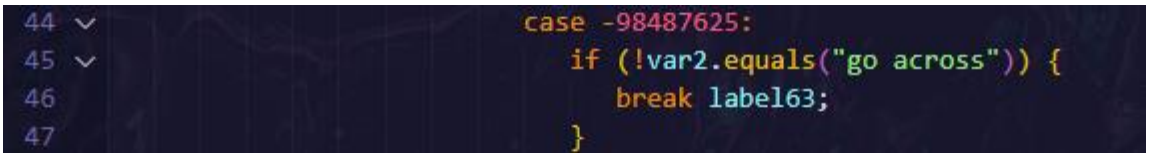
Notable command:

```
Take rope
```

Finding previous room:

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "CrystalRoom"
grep: Bridge.class: binary file matches
grep: CrystalRoom.class: binary file matches
```

Bridge.class:



```
                case -98487625:
                   if (!var2.equals("go across")) {
                      break label63;
                   }
```

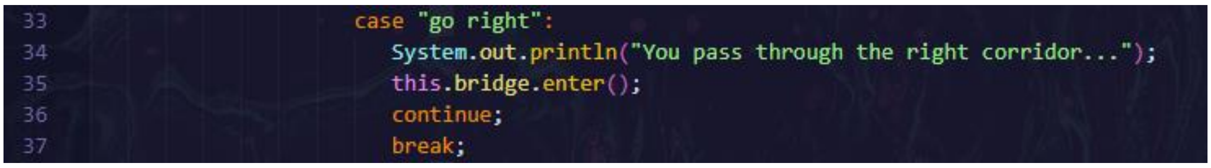
Notable command:

```
Go across
```

Previous room:

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "Bridge"
grep: Bridge.class: binary file matches
```

```
grep: EntryHall.class: binary file matches
```



```
                        case "go right":
                            System.out.println("You pass through the right corridor...");
                            this.bridge.enter();
                            continue;
                            break;
```

Taking the right path will redirect to bridge class. To combine the code of taking the rope will be:

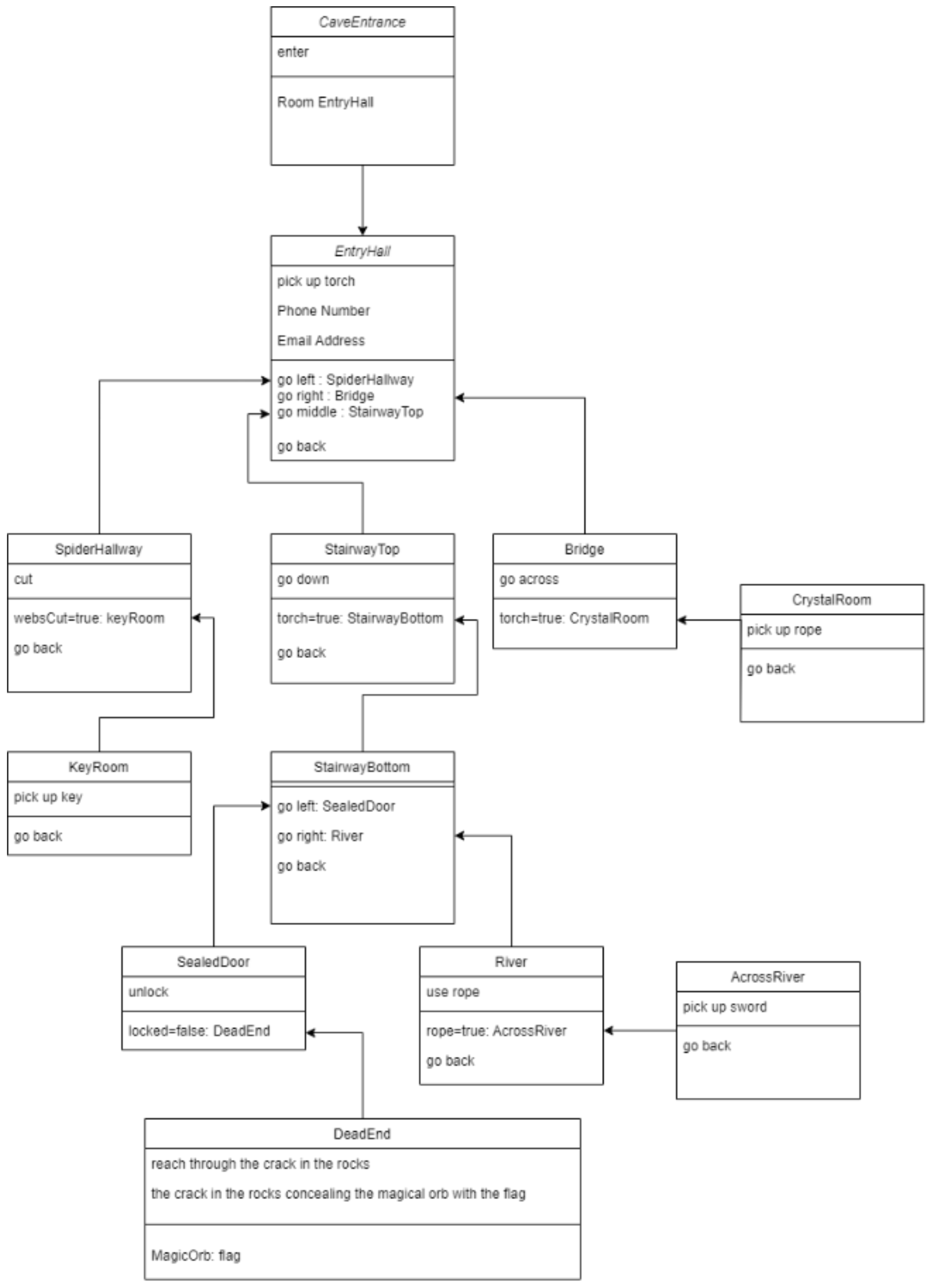
```
Go right
go across
pick up rope
```

Now what left is torch.

```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure/extract/rooms]
└─$ grep -r "pick up torch"
grep: EntryHall.class: binary file matches
```

Great pick up torch was in entryhall itself.

Now we understand the flow of the code will be:



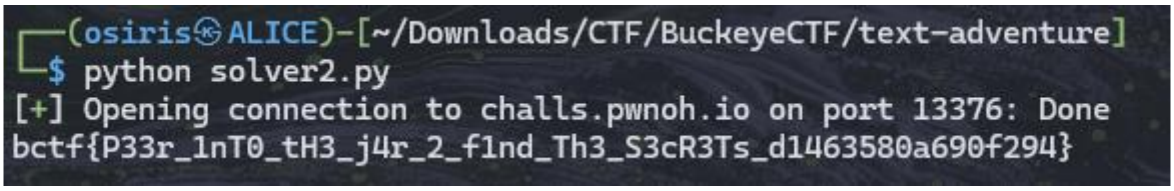
Command combine:

```
enter
pick up torch
go right
go across
pick up rope
go back
go back
go middle
go down
go right
use rope
pick up sword
go back
go back
go back
go back
go left
use sword
cut
pick up key
go back
go back
go middle
go down
go left
unlock
reach through the crack in the rocks
the crack in the rocks concealing the magical orb with the flag
```

Code (Python)

```python
from pwn import *
p = remote('challs.pwnoh.io', 13376)

p.sendline(b"enter")
p.sendline(b"pick up torch")
p.sendline(b"go right")
p.sendline(b"go across")
p.sendline(b"pick up rope")
p.sendline(b"go back")
p.sendline(b"go back")
p.sendline(b"go middle")
p.sendline(b"go down")
p.sendline(b"go right")
p.sendline(b"use rope")
p.sendline(b"pick up armor")
p.sendline(b"pick up sword")
p.sendline(b"take sword")
p.sendline(b"go back")
p.sendline(b"go back")
p.sendline(b"go back")
p.sendline(b"go back")
```

```
p.sendline(b"go left")
p.sendline(b"use sword")
p.sendline(b"cut")
p.sendline(b"pick up key")
p.sendline(b"go back")
p.sendline(b"go back")
p.sendline(b"go middle")
p.sendline(b"go down")
p.sendline(b"go left")
p.sendline(b"unlock")
p.sendline(b"reach through the crack in the rocks")
p.sendline(b"the crack in the rocks concealing the magical orb with the
flag")

while True:
    try:
        output = p.recvline().decode()
        if "bctf{" in output:
            print(output)
            break
    except Exception:
        pass

p.close()
```



```
┌──(osiris㉿ALICE)-[~/Downloads/CTF/BuckeyeCTF/text-adventure]
└─$ python solver2.py
[+] Opening connection to challs.pwnoh.io on port 13376: Done
bctf{P33r_1nT0_tH3_j4r_2_f1nd_Th3_S3cR3Ts_d1463580a690f294}
```

Flag: bctf{P33r_1nT0_tH3_j4r_2_f1nd_Th3_S3cR3Ts_d1463580a690f294}