1) Using hbv7.1jar.

There are two players in this game, Harry (a hider) and Sally (a seeker). The game begins with the hider Harry randomly hiding an object at any location in a 3x3 grid. Sally then walks through the grid starting from the location (0,0), following the red arrows shown in figure1 until she finds the hidden object (and Harry will say to Sally "Huckle buckle beanstalk!"). After each Sally's new location, Harry tells her whether she is 'hot' or 'warm' or 'cold'. In a 3x3 grid case - 'Hot' means she is at a location that is either directly North, South, West or East from the hidden object. 'Warm' means she is at a location that is directly diagonal (North-West, North-East, South-West and South-East) from the hidden object. 'Cold' means she is at a location that is neither hot nor warm.

However according to this program, the greater the location number that Harry hides the object, the longer it takes for Sally to find it. This is because no matter what Harry says to Sally, she will always walk through the grids from location 1 to location 9 (in an order) until she finds the object.

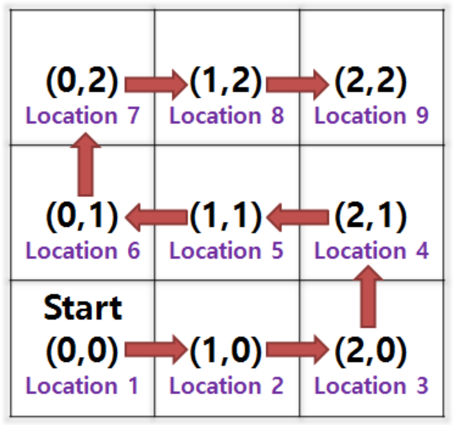
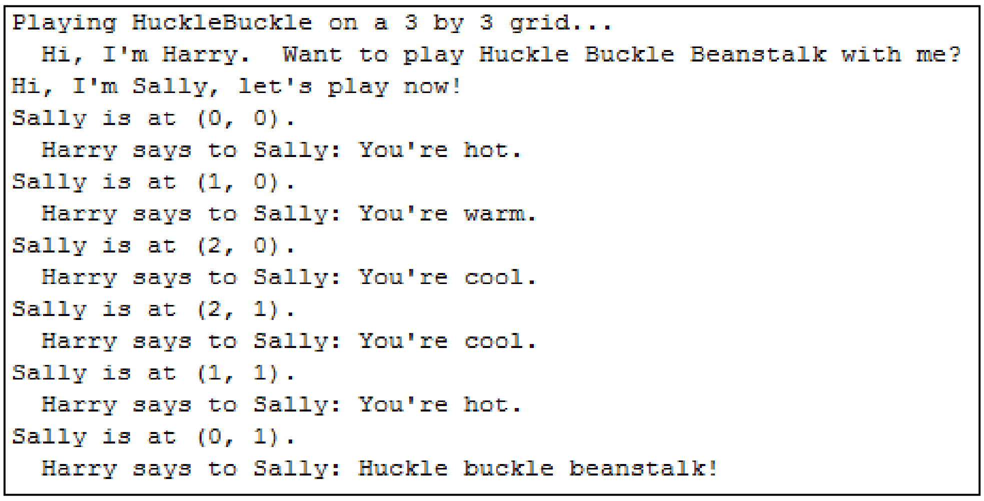


Figure1. A 3x3 grid shown with arrows and location numbers.

Example:

As you can see in figure2, Sally starts from location 1 (0,0) and walks to the next location indicated by the red arrows (figure1). In this example, Sally took 5 locations to find the hidden object because the hidden location was in location 6 (0,1).

For each of my other four observations, Sally took 3, 0, 2 and 4 turns to find the object respectively.



```
Playing HuckleBuckle on a 3 by 3 grid...
  Hi, I'm Harry.  Want to play Huckle Buckle Beanstalk with me?
Hi, I'm Sally, let's play now!
Sally is at (0, 0).
  Harry says to Sally: You're hot.
Sally is at (1, 0).
  Harry says to Sally: You're warm.
Sally is at (2, 0).
  Harry says to Sally: You're cool.
Sally is at (2, 1).
  Harry says to Sally: You're cool.
Sally is at (1, 1).
  Harry says to Sally: You're hot.
Sally is at (0, 1).
  Harry says to Sally: Huckle buckle beanstalk!
```

Figure2. A screenshot of a sample output from the runs of hbv7.1.

2) Modified main() method

```
public static void main(String[] args) {

    int gridSize = 3; // default value, if no args
    String seekerName = "";

    if( args.length > 2) {
        // third or more args: gives error
        System.err.println("Usage: hbv8 [gridSize[yourName]]");
        System.exit(1);
    } else {
        // first arg: gridSize
        if (args.length > 0) {
            try {
                gridSize = Integer.parseInt(args[0]);
            } catch (NumberFormatException e) {
                System.err.println("Usage: hbv8 [gridSize]");
                System.exit(1);
            }
            if(gridSize < 1 || gridSize > 40) {
                System.err.println("Error: gridSize must be in the range 1..40.  ");
                System.exit(1);
            }
        }
        // second arg: seekerName
        if( args.length > 1) {
            try {
                seekerName = args[1];
            } catch (Exception e) {
                System.err.println("Usage: hbv8 [gridSize]");
                System.exit(1);
            }
        }
    }

    Game myGame = new Game( gridSize, seekerName );
    myGame.play();
}
```

Modified constructor for Game

```
class Game {

    Hider myHider;
    Seeker mySeeker;
    int gridSize;

    Game( int gs, String name ) {
        gridSize = gs;
        System.out.println( "Playing HuckleBuckle on a " + gs + " by " + gs + " grid..." );
        myHider = new Hider( "Harry", this );

        if (name.equals("")){
            mySeeker = new Seeker( "Sally", this );
        } else {
            mySeeker = new Seeker(name, this);
        }
    }

    void play() {
        mySeeker.seek();
    }
}
```

3) HumanSeeker class

```java
package hucklebuckle;

import java.util.Scanner;

class HumanSeeker extends Seeker{

    Scanner input;

    HumanSeeker(String n, Game g){
        super(n,g);
        input = new Scanner(System.in);
    }

    void seek() {
        System.out.println( "Hi, I'm " + getName() + ", I'm ready to play!  " );
        System.out.println(reportLocation());

        while (game.myHider.revealTemperature(this) != Temperature.HUCKLEBUCKLEBEANSTALK){
            System.out.print("Please type a directional character (n, s, e, w) OR type q to give up: ");
            String userInput = input.next();
            char c = userInput.charAt(0);
            if (c == 'n'){
                this.move(0, 1);
            } else if (c == 's'){
                this.move(0, -1);
            } else if (c == 'e'){
                this.move(1, 0);
            } else if (c == 'w'){
                this.move(-1, 0);
            } else if (c == 'q'){
                System.out.println("I give up. I can't find you!");
                System.exit(0);
            }
        }
    }

}
```

Modified a method in Game class

```java
class Game {

    Hider myHider;
    Seeker mySeeker;
    HumanSeeker myHumanSeeker;
    int gridSize;
    boolean humanSeeker = false;

    Game( int gs, String name ) {
        gridSize = gs;
        System.out.println( "Playing HuckleBuckle on a " + gs + " by " + gs + " grid..." );
        myHider = new Hider( "Harry", this );

        if (name.equals("")){
            mySeeker = new Seeker( "Sally", this );
        } else {
            myHumanSeeker = new HumanSeeker(name, this);
            humanSeeker = true;
        }
    }

    void play() {
        if (!humanSeeker){
            mySeeker.seek();
        } else {
            myHumanSeeker.seek();
        }
    }
}
```

I've created and initialised a boolean variable called humanSeeker to false. This means that we are not in control of the seeker instance (in fact, the computer seeker "Sally" will be the seeker instance). However, if we supply a second commandline argument, this boolean variable will be changed to true. Therefore when play() method is called from HuckleBuckleclass, we can access to different seek() methods depending on the value of the humanSeeker. i.e. If humanSeeker is false, then mySeeker.seek(); method will be called, and if humanSeeker is true, myHumanSeeker.seek() method will be called.

4) <u>HumanSeeker class</u>

```
package hucklebuckle;

import java.awt.Point;
import java.util.Scanner;

class HumanSeeker extends Player{

    Scanner input;

    HumanSeeker(String n, Game g){
        super(new Point(0,0), n, g);
        input = new Scanner(System.in);
    }

    void seek() {
        System.out.println( "Hi, I'm " + getName() + ", I'm ready to play!  " );
        System.out.println(reportLocation());

        while (game.myHider.revealTemperature(this) != Temperature.HUCKLEBUCKLEBEANSTALK){
            System.out.print("Please type a directional character (n, s, e, w) OR type q to give up: ");
            String userInput = input.next();
            char c = userInput.charAt(0);
            if (c == 'n'){
                this.move(0, 1);
            } else if (c == 's'){
                this.move(0, -1);
            } else if (c == 'e'){
                this.move(1, 0);
            } else if (c == 'w'){
                this.move(-1, 0);
            } else if (c == 'q'){
                System.out.println("I give up. I can't find you!");
                System.exit(0);
            }
        }
    }

}
```

The only difficulty while implementing this design was that I had to modify the revealTemperature() method as well. I had an error in the while loop in the seek() method in the HumanSeeker class. This was because the revealTemperature() method in the Hider class only received a parameter from the Seeker class (Seeker seeker).

In order to solve this problem, I copied the whole revealTemperature() method and created a new one for the HumanSeeker class, so that it could also receive a parameter from the HumanSeeker class (HumanSeeker humanSeeker).

5) HumanSeeker class

```java
package hucklebuckle;

import java.util.Scanner;

class HumanSeeker extends Seeker{

    Scanner input;

    HumanSeeker(String n, Game g){
        super(n,g);
        input = new Scanner(System.in);
    }

    void seek() {
        System.out.println( "Hi, I'm " + getName() + ", I'm ready to play!  " );
        System.out.println(reportLocation());

        while (game.myHider.revealTemperature(this) != Temperature.HUCKLEBUCKLEBEANSTALK){
            System.out.print("Please type a directional character (n, s, e, w) OR type q to give up: ");
            String userInput = input.next();
            char c = userInput.charAt(0);
            if (c == 'n'){
                this.move(0, 1);
            } else if (c == 's'){
                this.move(0, -1);
            } else if (c == 'e'){
                this.move(1, 0);
            } else if (c == 'w'){
                this.move(-1, 0);
            } else if (c == 'q'){
                System.out.println("I give up. I can't find you!");
                System.exit(0);
            }
        }
    }

}
```

I actually found no difficulties at all.

I just had to make the Seeker class into an abstract class with two subclasses, HumanSeeker and ComputerSeeker. Because "The ComputerSeeker should have the same behaviour as the Seeker in hbv7.1", I copied the seek() method from the Seeker in hbv7.1 and pasted into the ComputerSeeker class. And then I modified the Game class slightly. Overall, the third design option was relatively easy.

6)

Let's have a look at some of the features of each design option:

First design option (HumanSeeker inherit from Seeker).

- HumanSeeker is a subclass of Seeker (poorly structured, a bit weird).
- Difficult to read and understand.
- Relatively hard to modify and develop

Second design option (HumanSeeker inherit from Player)

- Good structure.
- HumanSeeker and ComputerSeeker are at the same level.
- Easy to read and understand.
- Easy to modify and develop.

Third design option (Seeker be an abstract class with two subclasses, HumanSeeker and ComputerSeeker)

- Very well structured in terms of the inheritance of the classes.
- HumanSeeker and ComputerSeeker are subclasses of the Seeker at the same level. This will result in Player class having two distinct subclasses, Hider and Seeker (this is good, because any player will be either hider or seeker).
- Very easy to read and understand.
- Very easy to modify and develop.

Overall, the **third design option** is the **best choice** for this program.