DD1341 Introduktion till datalogi 2010/2011

Uppgift nummer: ......

Namn: ....

Grupp nummer: ......

Övningsledare: ....

Betyg: .....   Datum: ....   Rättad av: ....

Innehåll

World of Zuul

Karaktärer 7.48 och 7.49

Karaktärer finns implementerade i Character-klassen på sida 6. För att prata med dem används CommandTalk-klassen på sida 14. Det finns en inställning på karaktärerna som säger ifall de ska runt slumpvist i spelet och den finns i Game-klassen på sida 19 rad 215.

Modulär kod 7.47

Varje kommando har en egen klass där man åsidosätter executeCommand-metoden för att låta den utföra kommandot. Så för att lägga till ett nytt kommando skapar man en ny klass som ärver CommandWord och sen lägger man till klassen i parsern så den vet om att det nya kommandot finns. Varje karaktär har också en egen klass och ifall karaktären ska göra mer advancerade saker kan de åsidosätta en del av de metoder som finns. Den nya metoden kommer då att köras istället för orginalmethoden och gör att karaktären fungerar som man själv vill. Samma sak gäller för föremål.

Trapdoor 7.43

För att skapa en fälla finns det en en dörr som går till ett rum där det sedan inte finns någon utgång tillbaka ifrån. Fällan skapas i Game-klassen på sida 19 rad 97.

Teleport 7.46

Man kan teleportera sig tillbaka till det första rummet genom att använda föremålet ”helig stensom finns i Items-enumen på sidan 27 rad 28. Det skulle vara lätt att låta den slumpa vilket rum man skulle hamna i genom att hämta ut ett slumpvist rum från kartan istället. Men för att underlätta i spelet kommer man alltid till första rummet igen så man kan ta sig ut.

Teleport 7.45

Det finns låsta dörrar. De kan låsas upp genom att hitta rätt föremål och sedan använda CommandUnlock (sida 15) på dörren.

Övriga roliga saker

Det går att spara och ladda spel. Finns i slutet av Game-klassen på sida 19 rad 300 och 318.

Jag gjorde min egen terminal med swingkomponenter, vilken använder sig av *System.in* och *System.out* som sedan länkas till en textarea och en textfield. Den har historik för kommandon för att underlätta spelandet det finns i klassen Console på sida 38 och i klassen ConsoleGUI på sida pagerefconsolegui.

Skriver man 'hjälp' på något kommando som inte finns hämtar den ut en beskrivning från wikipedia. Så skriver man 'hjälp dörr' hämtar den ut vad som står om dörrar på wikipedia. Det finns i CommandHelp-klassen på sida 9 rad 59.

Källkod

Beggar

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Beggar.java

```java
package org.x2d.zuul;
/**
 * A beggar NPC. He wants food and will give the player a key to the
    south door
 * when he gets the food.
 */
public class Beggar extends Character {
    private boolean gotFood = false;
    private int tCounter1 = 0;
    private int tCounter2 = 0;
    /**
     * Creates a new beggar NPC.
     */
    public Beggar() {
        super(Game.generateName(), "Kan du skänka mat till en hungrig
            stackare?");
    }

    @Override
    public void talk(Game g, Player p) {
        //When the player has given him food
        if (gotFood) {
            String[] texts = {
                "Underbart med mat!",
                "Nom, nom, nom!",
            };
            System.out.println(texts[tCounter1]);
            tCounter1 = (tCounter1 + 1) % texts.length;
            return;
        }
        //If the player has no food reply some text
        if (!p.hasItem(Items.FOOD)) {
            String[] texts = {
                "Kan du skänka mat till en hungrig stackare?",
                "Jag är så hungrig.",
                "Snälla kan jag få lite mat?",
                "Ingen tänker på stackars lilla mig."
            };
            System.out.println(texts[tCounter1]);
            tCounter1 = (tCounter1 + 1) % texts.length;
        //If the player has food give him the key and remove the food
        } else {
            System.out.println("Å, mat till mig? Jag har inte ätit på en
                vecka!");
            System.out.println("Jag ska hjälpa dig att öppna den södra
                dörren.");
            System.out.println("Här får du nyckeln, jag hitta den på
                marken här utanför.");
            p.removeItem(Items.FOOD);
            p.addItem(Items.STORE_ROOM_KEY);
            gotFood = true;
        }
    }
}
```

Cat

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Cat.java

```java
package org.x2d.zuul;
/**
 * A cat NPC that walks around randomly in the world and
 * really does nothing useful.
 */
public class Cat extends Character {
    private int tCounter1 = 0;
    /**
     * Creates a new cat NPC.
     */
    public Cat() {
        super("Katt", "Mjauu!");
        setWalkRandomly(true);
    }

    @Override
    public void talk(Game g, Player p) {
        String[] texts = {
            "Mjau",
            "Nom, nom, nomä!",
            "Mjaaaaaaaau!"
        };
        System.out.println(texts[tCounter1]);
        tCounter1 = (tCounter1 + 1) % texts.length;
        return;

    }
}
```

Character

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Character.java

```java
package org.x2d.zuul;
import java.io.*;
/**
 * Character class. Stores information about one NPC in the world.
 * Subclasses should override methodes when needed to create new
 * functionality for a NPC. Here comes and example of how it can be used:
<pre>
public class MyCharacter extends Character {
    public MyCharacter() {
        super("my char name", "something the character says");
    }
    &#0064;Override
    public void talk(Game g, Player p) {
        //Something that should happen when the player talks to this
        //character.
    }
}
</pre>
 */
public abstract class Character implements Serializable {
    private Room currentRoom;
    private String name;
    private String textFirstTime;
    private Item wantedItem;
    private boolean walkRandomly = false;
    private boolean isFirstTime = true;

    /**
     * Creates a new Character.
     * @param name The character's name.
     * @param firstTime A text that will be displayed the first time the
         player
     * meets this NPC.
     */
    public Character(String name, String firstTime) {
        this.name = name;
        this.textFirstTime = firstTime;
    }

    /**
     * Method that contains all the dialog and special things that should
     * happen when the player talks with this NPC
     *
     * @param g The game.
     * @param p The player.
     */
    public abstract void talk(Game g, Player p);

    /**
     * Gets the room which the NPC is in.
     *
     * @return The room.
     */
    public Room getCurrentRoom() {
        return currentRoom;
    }

    /**
     * Sets the room which the character is in.
     *
     * @param room The room.
     */
    public void setCurrentRoom(Room room) {
        currentRoom = room;
    }

    /**
     * Gets the character's name.
     *
     * @return The character's name.
     */
    public String getName() {
        return name;
    }

    /**
     * Gets the item the character wants. If he retrives this item
         something will happen.
     *
     * @return The item the character want or <code>null</code> if the
         character doesn't want an item.
     */
    public Item getWantedItem() {
        return wantedItem;
    }

    /**
     * Sets the item the character wants. If he retrives this item
         something will happen.
     *
     * @param wantedItem Item that the character want.
     */
    public void setWantedItem(Item wantedItem) {
        this.wantedItem = wantedItem;
    }

    /**
     * Sets if the character should walk around on the map.
     *
     * @param isWalkingRandomly <code>true</code> if the character should
          walk around on the map else <code>false</code>.
     */
    public void setWalkRandomly(boolean isWalkingRandomly) {
        walkRandomly = isWalkingRandomly;
    }

    /**
     * Gets if the character should walk around on the map.
     *
     * @return <code>true</code> if the character should walk around on
         the map else <code>false</code>.
     */
    public boolean isWalkingRandomly() {
        return walkRandomly;
    }

    /**
     * Gets the text that will be displayed the first time the player
         meets this npc.
     *
     * @return The text that will be displayed the first time the player
         meets this npc.
     */
    public String getFirstTimeText() {
        setFirstTime(false);
        return textFirstTime;
    }

    /**
     * Gets if this is the first time the player meets this NPC.
     *
     * @return <code>true</code> if this is the first time else <code>
         false</code>.
     */
    public boolean isFirstTime() {
        return isFirstTime;
    }

    /**
     * Sets if this is the first time the player meets this NPC.
     *
     * @param isFirstTime <code>true</code> if the character should walk
         around on the map else <code>false</code>.
     */
    public void setFirstTime(boolean isFirstTime) {
        this.isFirstTime = isFirstTime;
    }
}
```

CommandGo

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandGo.java

```
package org.x2d.zuul;
/**
 * Command word go. This command tries to move the character in the
     chosen direction.
 */
public class CommandGo extends CommandWord {
    public CommandGo(String commandName) {
        super(commandName);
    }

    @Override
    public String[] getTargets(Game game) {
        return game.getPlayer().getCurrentRoom().getExits();
    }

    /**
     * @param target The direction the player should go.
     */
    @Override
    public void executeCommand(Game game, String target) {
        if (game.getPlayer().getCurrentRoom().getDoor(target)==null) {
            System.out.println("Det finns inget utgång åt det hållet.");
            return;
        }
        game.getPlayer().goRoom(target);
        if (game.getPlayer().getCurrentRoom() == game.getRoom(Game.Rooms.
            OUTDOOR)) {
            game.endGame(true);
        }
    }

    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandHelp

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandHelp.java

```
package org.x2d.zuul;
import java.util.*;
import java.io.*;
import java.net.*;
import java.util.regex.*;
/**
 * Command word help. This command word lists all possible command words
    which can be used at the current time.
 * If another command word is used as target then a list of possible
    targets for that command word will be displayed.
 */
public class CommandHelp extends CommandWord {
    public CommandHelp(String commandName) {
        super(commandName);
    }
    @Override
    public String[] getTargets(Game game) {
        Collection<CommandWord> commands = game.getParser().
            getCommandWords();
        String[] tmpArray = new String[commands.size()];
        int i=0;
        for (CommandWord cw : commands) {
            tmpArray[i++] = cw.getCommand();
        }
        return tmpArray;
    }

    /**
     * @param target If <code>null</code> then it lists all possible
        command words which can be used at the current time
     * else if another command word is used as target then a list of
        possible targets for that command word will be displayed.
     * If the there is a target but there is no command word with that
        name, then it will try to check wikipedia for more information.
     */
```

```java
    @Override
    public void executeCommand(Game game, String target) {
        Parser parser = game.getParser();
        if (target==null) {
            System.out.println("Följande saker kan du göra: ");
            Collection<CommandWord> cWordsCollection = parser.getCommandWords();
            for (CommandWord cw : cWordsCollection) {
                if (!cw.mustHaveTarget()||cw.getTargets(game).length!=0) {
                    System.out.print(cw.getCommand()+" ");
                }
            }
        } else if (!parser.isCommand(target)) {
            //If the command word which help is requested for does not exist:
            //search for some information on wikipedia.
            printWikipedia(target);
        } else {
            String[] targets = parser.getCommand(target).getTargets(game);
            if (targets.length==0) {
                System.out.println("Det finns inga möjliga mål för: "+target);
            } else {
                System.out.println(String.format("Möjliga mål för kommandot %s är följande: ", target));
                for (String s : targets) {
                       System.out.print(s+" ");
                }
            }
        }
    }

    //Tries to find some information from wikipedia from the search string.
    private void printWikipedia(String search) {
        try {
            URL wiki = new URL("http://sv.wikipedia.org/wiki/"+URLEncoder.encode(search, "UTF-8"));
            Scanner reader = new Scanner(new InputStreamReader(wiki.openStream()));
            reader.findWithinHorizon(Pattern.compile("<p>(.{0,10})<b>"), 0);
            reader.useDelimiter("(</p>)");
            //Prints two paragraphs and removes html-tags and white-space chars.
            for (int i=0;i<2;i++) {
                System.out.println(reader.next().replaceAll(("<.*?>"), "").replaceAll("((?<=\\s)(\\s+))", "").trim());
            }
            reader.close();
        } catch (Exception e) {
            System.out.println("Inte ens wikipedia förstår vad du vill ha hjälp med.");
        }
    }

}
```

CommandList

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandList.java

```java
package org.x2d.zuul;
import java.util.*;
```

```java
/**
 * Command word list items. Lists all the items in the player's backpack.
 */
public class CommandList extends CommandWord {
    public CommandList(String commandName) {
        super(commandName);
    }

    /**
     * @param target Never used.
     */
    @Override
    public void executeCommand(Game game, String target) {
        System.out.println("Följande saker finns i din ryggsäck: ");
        Collection<Items> pItems = game.getPlayer().getItems();
        if (pItems.size()==0) {
            System.out.println(" - Inga");
            return;
        }
        for (Items item : pItems) {
            Item itemObject = item.getItem();
            System.out.println(" - "+itemObject.getName()+": "+itemObject
                .getDescription());
        }
    }

    @Override
    public boolean mustHaveTarget() {
        return false;
    }
}
```

CommandLoad

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandLoad.java

```java
package org.x2d.zuul;
import java.io.*;
/**
 * Command word load. Loads a previously saved game. It tries to load the
      file from the 'saves' folder in the game folder.
 */
public class CommandLoad extends CommandWord {
    private File saveGameDir;
    /**
     * Constructor.
     * @param saveDir Sets the dir where save games should be saved.
     */
    public CommandLoad(String commandName, File saveDir) {
        super(commandName);
        saveGameDir = saveDir;
    }
    public CommandLoad(String commandName) {
        this(commandName, new File("saves/"));
    }

    @Override
    public String[] getTargets(Game game) {
        if (!saveGameDir.exists()) {
            return null;
        }
        //Lists all files in 'saves/' ending with '.zul'
        File[] files = (new File("saves/")).listFiles(new SaveGameFilter
            ());
        String[] fileNames = new String[files.length];
        for (int i=0;i<files.length;i++) {
            String fileName = files[i].getName();
            fileNames[i] = fileName.substring(0, fileName.length()-4);
        }
        return fileNames;
    }

    /**
     * @param target The name of the file which should be loaded.
     */
    @Override
    public void executeCommand(Game game, String target) {
        if (target==null || target == "") {
            System.out.println("Du måste skriva in namnet på filen du vill ladda.");
            return;
        }
        target = target.replaceAll("([^\\w])", ""); //Makes sure only [a-ö,0-9] are used.
        File f = new File("saves/"+target+SaveGameFilter.SAVE_GAME_EXTENSION);
        if (!f.exists()) {
            System.out.println("Filen du försöker ladda finns inte: "+ f);
        }
        game.loadGame(f);
        System.out.println("Ditt spel har blivit laddat.");
        System.out.println(game.getPlayer().getCurrentRoom().getLongDescription());
    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandRead

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandRead.java

```java
package org.x2d.zuul;
import java.util.*;
/**
 * Command word read. Used to read items which isReadable() returns true.
 * The target paramenter should be the name of an item either in the
 * player's backpack or in the room.
 */
public class CommandRead extends CommandWord {
    public CommandRead(String commandName) {
        super(commandName);
    }

    @Override
    public String[] getTargets(Game game) {
        //Checks both the player's backpack and the current room.
        Collection<Items> pItems = game.getPlayer().getItems();
        Collection<Items> rItems = game.getPlayer().getCurrentRoom().getItems();
        ArrayList<String> items = new ArrayList<String>();
        checkReadable(items, pItems);
        checkReadable(items, rItems);
        return items.toArray(new String[0]);
    }

    //getTarget help method.
    private void checkReadable(ArrayList<String> items, Collection<Items>
         itemCollection) {
        for (Items i : itemCollection) {
            if (i.getItem().isReadable()) {
                items.add(i.getItem().getName());
            }
        }
    }

    /**
     * @param target The name of the item which the player should try to
         read from.
     */
    @Override
    public void executeCommand(Game game, String target) {
        Items item = Items.getItem(target);
        if (!game.getPlayer().hasItem(item)) {
            if (!game.getPlayer().getCurrentRoom().hasItem(item)) {
                System.out.println(String.format("Det finns inget föremål
                     med namnet '%s' i det här rummet eller i din ryggsäck
                    .", target));
                return;
            }
        }
        if (!item.getItem().isReadable()) {
            System.out.println("Det finns inget att läsa på: "+target);
            return;
        }
        System.out.println("Följande står skrivet:");
        System.out.println(item.getItem().getText());

    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandSave

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandSave.java

```
package org.x2d.zuul;
import java.io.*;
/**
 * Command word save. Saves the current game.
 */
public class CommandSave extends CommandWord {
    private File saveGameDir;
    /**
     * Constructor.
     * @param saveDir Sets the dir where save games should be saved.
     */
    public CommandSave(String commandName, File saveDir) {
        super(commandName);
        saveGameDir = saveDir;
    }
    public CommandSave(String commandName) {
        this(commandName, new File("saves/"));
    }

    @Override
    public String[] getTargets(Game game) {
        return new String[]{"Ett namn du vill använda för att spara ditt
            spel."};
    }

    /**
     * @param target The name of the file which the current game should
        be saved to.
     */
    @Override
    public void executeCommand(Game game, String target) {
        target = target.replaceAll("([^\\w])", ""); //Makes sure only [a-
            ö,0-9] are used.
        if (target==null || target == "") {
            System.out.println("Du måste välja ett namn på filen du vill
                spara.");
            return;
        }
        //If the save game folder does not exist create the needed
            folders.
        if (!saveGameDir.exists()) {
            saveGameDir.mkdirs();
        }
        game.saveGame(new File(saveGameDir, target+SaveGameFilter.
            SAVE_GAME_EXTENSION));
        System.out.println(String.format("Ditt spel har blivit sparat som
             '%s'.", target));
    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }
}
```

CommandTake

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandTake.java

```
package org.x2d.zuul;
import java.util.*;
/**
 * Command word take. This command is used when the player is trying to
    take an item and put it into
 * the backpack.
 */
public class CommandTake extends CommandWord {
    public CommandTake(String commandName) {
        super(commandName);
    }

    @Override
    public String[] getTargets(Game game) {
        Collection<Items> rItems = game.getPlayer().getCurrentRoom().
            getItems();
        ArrayList<String> items = new ArrayList<String>();
        for (Items i : rItems) {
            if (i.getItem().isTakable()) {
                items.add(i.getItem().getName());
            }
        }
        return items.toArray(new String[0]);
    }

    /**
     * @param target The name of the item which the player should try to
        take.
     */
    @Override
    public void executeCommand(Game game, String target) {
```

```java
        Items item = Items.getItem(target);
        if (!game.getPlayer().getCurrentRoom().hasItem(item)) {
            System.out.println(String.format("Det finns inget föremål med namnet '%s' i det här rummet.", target));
            return;
        }
        if (!item.getItem().isTakable()) {
            System.out.println("Du kan inte ta med dig: "+target);
            return;
        }
        game.getPlayer().getCurrentRoom().removeItem(item);
        game.getPlayer().addItem(item);
    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandTalk

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandTalk.java

```java
package org.x2d.zuul;
import java.util.*;
/**
 * Command word talk. This command is used when the player is trying to talk to a character.
 */
public class CommandTalk extends CommandWord {
    public CommandTalk(String commandName) {
        super(commandName);
    }

    @Override
    public String[] getTargets(Game game) {
        Collection<Character> rChars = game.getPlayer().getCurrentRoom().getCharacters();
        ArrayList<String> chars = new ArrayList<String>();
        for (Character cha : rChars) {
            chars.add(cha.getName());
        }
        return chars.toArray(new String[0]);
    }

    /**
     * @param target The name of the character which the player should try to talk to.
     */
    @Override
    public void executeCommand(Game game, String target) {
        Character cha = game.getPlayer().getCurrentRoom().getCharacter(target);
        if (cha == null) {
            System.out.println(String.format("Det finns ingen med namnet '%s' i det här rummet.", target));
            return;
        }
        System.out.println(String.format("Du börjar prata med %s.", cha.getName()));
        cha.talk(game, game.getPlayer());
    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandUnlock

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandUnlock.java

```java
package org.x2d.zuul;
import java.util.*;
/**
 * Command word unlock. This command is used when the player is trying to
      unlock a door.
 */
public class CommandUnlock extends CommandWord {
    public CommandUnlock(String commandName) {
        super(commandName);
    }

   @Override
    public String[] getTargets(Game game) {
        Room cRoom = game.getPlayer().getCurrentRoom();
        String[] exits = cRoom.getExits();
        ArrayList<String> tmpArray = new ArrayList<String>();
        for (String exit : exits) {
            Door door = cRoom.getDoor(exit);
            Items unlockItem = door.getUnlockItem();
            if (door.isLocked()&&unlockItem!=null&&game.getPlayer().
                hasItem(unlockItem)) {
                tmpArray.add(exit);
            }
        }
        return tmpArray.toArray(new String[0]);
    }

    /**
     * @param target The name of the direction which the player should
          try to unlock a door in.
     */
    @Override
    public void executeCommand(Game game, String target) {
        Door door = game.getPlayer().getCurrentRoom().getDoor(target);
        if (door==null) {
            System.out.println("Det finns inget att låsa upp åt "+target)
                ;
            return;
        }
        System.out.println(String.format("Dörren i %s har blivit upplåst"
            , target));
        door.setLocked(false);
    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandUse

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandUse.java

```java
package org.x2d.zuul;
import java.util.*;
/**
```

```
 * Command word use. This command is used when the player is trying to
     use an item.
 */
public class CommandUse extends CommandWord {
    public CommandUse(String commandName) {
        super(commandName);
    }

    @Override
    public String[] getTargets(Game game) {
        Collection<Items> pItems = game.getPlayer().getItems();
        Collection<Items> rItems = game.getPlayer().getCurrentRoom().
            getItems();
        ArrayList<String> items = new ArrayList<String>();
        checkUsable(items, pItems);
        checkUsable(items, rItems);
        return items.toArray(new String[0]);
    }

    //getTarget help method.
    private void checkUsable(ArrayList<String> items, Collection<Items>
        itemCollection) {
        for (Items i : itemCollection) {
            if (i.getItem().isUsable()) {
                items.add(i.getItem().getName());
            }
        }
    }
    /**
     * @param target The name of the item which the player should try to
         use.
     */
    @Override
    public void executeCommand(Game game, String target) {
        Items item = Items.getItem(target);
        if (game.getPlayer().getCurrentRoom().hasItem(item)) {
            if (game.getPlayer().hasItem(item)) {
                System.out.println(String.format("Det finns inget föremål
                     med namnet '%s' i det här rummet eller i din ryggsäck
                    .", target));
                return;
            }
        }
        if (!item.getItem().isUsable()) {
            System.out.println("Det går inte att använda "+target);
            return;
        }
        item.getItem().use(game);

    }
    @Override
    public boolean mustHaveTarget() {
        return true;
    }

}
```

CommandWord

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/CommandWord.java

```
package org.x2d.zuul;
/**
 * Command word class. This class should be used as a super class for
 * all command words. There are two kinds of commands. The first type
     does not need
 * a target. These commands only need to override the executeCommand(
     String target) method.
 * The second type does need a target and then the executeCommand(String
     target), mustHaveTarget()
 * and getTargets() must be overriden. Here comes an example:
<pre>
public class CommandMyCommand extends CommandWord {
    public CommandMyCommand(String commandName) {
        super(commandName);
    }
    &#0064;Override
    public String[] getTargets(Game game) {
        return new String[0]; //Should be a list of possible targets
    }
    &#0064;Override
    public void executeCommand(Game game, String target) {
        //Something that should happen when this command word is used
    }
}
</pre>
 */
public abstract class CommandWord
{
    // instance variables - replace the example below with your own
    private String command;

    /**
     * Constructor for objects of class CommandWord
     *
     * @param command The command.
     */
    public CommandWord(String command)
    {
        if (!command.matches("[\\wåäö]+")) {
            throw new IllegalArgumentException("A command word can only
                use [a-ö,0-9]");
        }
        this.command = command;
    }

    /**
     * Gets the command string.
     *
     * @return The command string.
     */
    public String getCommand() {
        return command;
    }

    /**
     * Should be overriden. It's here the command gets executed.
     *
     * @param game The game.
     * @param target The target.
     */
    public abstract void executeCommand(Game game, String target);

    /**
     * Returns an array with all the possible targets at the current time
         .
     * This method should be overriden if the command needs a target.
     *
     * @return An array wih all the current possible targets for this
         command.
     */
    public String[] getTargets(Game game) {
            return new String[0];
        }

        /**
         * Should be overriden to say if this command needs a target to work.
         *
         * @return If this command must have a target.
         */
        public boolean mustHaveTarget() {
            return false;
        }
}
```

Door

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Door.java

```
package org.x2d.zuul;
import java.io.*;
/**
 * This class represents a door going between two rooms.
 */
public class Door implements Serializable
{
    private boolean isLocked;
    private Items unlockItem;
    private Room room1, room2;

    /**
     * Constructor for objects of class Door
     *
     * @param room1 The first room.
     * @param room2 The second room.
     * @param isLocked If the door is locked or not.
     */
    public Door(Room room1, Room room2, boolean isLocked)
    {
        this.room1 = room1;
        this.room2 = room2;
        this.isLocked = isLocked;
    }
    /**
     * Constructor for objects of class Door
     */
    public Door() {
        this(null, null, false);
    }

    /**
     * Constructor for objects of class Door
     *
     * @param isLocked If the door is locked or not.
     */
    public Door(boolean isLocked) {
        this(null, null, isLocked);
    }

    /**
     * Sets an item which is needed to unlock this door.
     *
     * @param item The item.
     */
    public void setUnlockItem(Items item) {
        unlockItem = item;
    }

    /**
     * Gets the item which is needed to unlock this door.
     *
     * @return item The item.
     */
    public Items getUnlockItem() {
        return unlockItem;
    }

    /**
     * Sets if this door should be locked or not.
     *
     * @param isLocked Should be <code>true</code> if the door should be
         locked else <code>false</code>.
     */
    public void setLocked(boolean isLocked) {
        this.isLocked = isLocked;
    }

    /**
     * Gets if this door should be locked or not.
     *
     * @return <code>true</code> if the door is locked else <code>false</
         code>.
     */
    public boolean isLocked() {
        return isLocked;
    }

    /**
     * Returns the exit room.
     *
     * @param entrance If this is room1, then room2 is returned, and if
         it's room2 then room1 is returned.
     * @return The exit room.
     */
    public Room getExit(Room entrance) {
        return (entrance==room1)?room2:room1;
    }

    /**
     * Sets the first room.
     *
     * @param r1 The first room.
     */
    public void setRoom1(Room r1) {
        this.room1 = r1;
    }

    /**
     * Sets the second room.
     *
     * @param r2 The second room.
     */
    public void setRoom2(Room r2) {
        this.room2 = r2;
    }

}
```

Game

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Game.java

```
package org.x2d.zuul;
import java.awt.*;
```

```java
import java.util.*;
import java.io.*;
import org.x2d.console.*;

/**
 *  This class is the main class of the "World of Zuul" application.
 *  "World of Zuul" is a very simple, text based adventure game.  Users
 *  can walk around some scenery. That's all. It should really be extended
 *  to make it more interesting!
 *
 *  To play this game, create an instance of this class and call the "play"
 *  method.
 *
 *  This main class creates and initialises all the others: it creates all
 *  rooms, creates the parser and starts the game.  It also evaluates and
 *  executes the commands that the parser returns.
 *
 * @author  Michael Kolling and David J. Barnes
 * @version 2008.03.30
 */
public class Game
{
    private Parser parser;
    private Player player;
    private boolean notFinished = true;
    private static HashSet<String> usedNames = new HashSet<String>();
    public static enum Rooms {
        OUTDOOR, ENTRANCE, CORRIDORE1, PIT, DINING_ROOM, KITCHEN, CORRIDORE2,
        CORRIDORE3, CORRIDORE4, STORE_ROOM, BRIDGE, GARDEN, TAVERN, TEMPLE, TELEPORT;
    }
    public static enum Directions {
        NORTH("norr"), SOUTH("söder"), WEST("väster"), EAST("öster");
        Directions(String value) {
            this.value = value;
        }
        private String value;
        public String getValue() {
            return value;
        }
    }
    private EnumMap<Rooms, Room> map = new EnumMap<Rooms, Room>(Rooms.class);
    private HashMap<String, Character> characters = new HashMap<String, Character>();

    /**
     * Starts a new game.
     */
    public static void main(String args[]) {
        Game g = new Game();
        g.play();
    }

    /**
     * Create the game and initialise its internal map.
     */
    public Game()
    {
        new ConsoleGUI();
        createParser();
        createPlayer();
        createRooms();
    }

    /**
     * Create all the rooms and link their exits together.
     */
    private void createRooms()
    {
        Room outdoor, entrance, corridore1, pit, diningRoom, kitchen,
            corridore2, corridore3, corridore4, storeRoom;
        Room bridge, garden, tavern, temple, teleport;
        Door tmpDoor;
        //Creates rooms
        map.put(Rooms.ENTRANCE, entrance = new Room("Det är ett mörkt och
             dystert rum, endast upplyst av några facklor. Golv och väggar
             är gjorda av stora massiva stenar. Du hör musik komma från
            den södra dörren."));
        map.put(Rooms.OUTDOOR, outdoor = new Room("Du är nu utanför
            borgen. Solen lyser och allt du vet är att du aldrig vill
            återvänta till den mörka borgen."));
        map.put(Rooms.CORRIDORE1, corridore1 = new Room("Du kommer in i
            en gång som forsätter så långt du kan se, in i själva berget.
            Ser ut att vara en gammal övergiven gruvgång. Det är helt
            mörkt längre in i gången."));
        map.put(Rooms.PIT, pit = new Room("Du såg inget i mörkret och
            ramla ner i ett gammalt gruvschakt. Du känner efter åt alla
            håll men du hittar bara solid sten."));
        map.put(Rooms.DINING_ROOM, diningRoom = new Room("Det är musik
            och rörelse i rummet. Det sitter 4 personer vid det ena bordet
             som ser ut som de inte vill bli störda. Det sitter en ensam
            man vid ett av de andra och äter."));
        map.put(Rooms.KITCHEN, kitchen = new Room("Du har kommit in i ett
             kök. Vilka det än var som lagade maten så är de inte kvar
            längre men du känner lukten av mat som de har lämnat kvar."));
        map.put(Rooms.CORRIDORE2, corridore2 = new Room("Du hör musik som
             kommer från den norra dörren."));
        map.put(Rooms.CORRIDORE3, corridore3 = new Room("Det ligger
            mängder av obetydliga saker på golvet som ser ut att ha ramlat
             av diverse transporter. Det ser ut som de har kommit eller
            gått från den västra utgången."));
        map.put(Rooms.CORRIDORE4, corridore4 = new Room("Mängder av skräp
             på golvet, utöver det finns inget av betydelse."));
        map.put(Rooms.STORE_ROOM, storeRoom = new Room("Nu förstår du vad
             allt skräp i de tidigare gången kom ifrån. Du har kommit in i
             en lagerlokal där det finns massor av lådor och tunnor
            staplade längs väggarna."));
        map.put(Rooms.BRIDGE, bridge = new Room("Du har kommit ut och
            står på en bro. Den går över en å som går långt nedanför bron.
             Du kan skymta träd och grönska söder ut och du ser borgen
            bakom dig."));
        map.put(Rooms.GARDEN, garden = new Room("Du står i en park med en
             fontän i mitten. Väster ut ser du ett tempel och öster ut ser
             du ett värdshus."));
        map.put(Rooms.TEMPLE, temple = new Room("Det är en lugn och tyst
            plats. Du har kommit in borgens tempel. Du känner att inget
            kan gå fel så länge du är inne i templet och att alla dina
            problem snart kommer att ordna sig."));
        map.put(Rooms.TELEPORT, teleport = new Room("Du kommer längre in
            i templet och ser ett blått sken lysa mot en sten i mitten av
            rummet."));
        map.put(Rooms.TAVERN, tavern = new Room("Du har kommit in i
            värdshuset. Det är en livlig miljö med massor av folk och
            ljudnivån är hög."));

        //Creates doors
        tmpDoor = new Door(true);
        tmpDoor.setUnlockItem(Items.STONE_OF_DELEN);
        entrance.setEntrance(Directions.NORTH.getValue(), outdoor,
            tmpDoor);
        entrance.setEntrance(Directions.WEST.getValue(), corridore1, new
            Door());
        entrance.setEntrance(Directions.SOUTH.getValue(), diningRoom, new
             Door());
        corridore1.setEntrance(Directions.WEST.getValue(), pit, new Door
            ());
        pit.setExit(Directions.EAST.getValue(), null);
        diningRoom.setEntrance(Directions.WEST.getValue(), kitchen, new
            Door());
        diningRoom.setEntrance(Directions.SOUTH.getValue(), corridore2,
            new Door());
        kitchen.setEntrance(Directions.WEST.getValue(), null, new Door(
            true));
        tmpDoor = new Door(true);
        tmpDoor.setUnlockItem(Items.STORE_ROOM_KEY);
        corridore2.setEntrance(Directions.SOUTH.getValue(), corridore3,
            tmpDoor);
        corridore3.setEntrance(Directions.WEST.getValue(), corridore4,
            new Door());
        corridore3.setEntrance(Directions.SOUTH.getValue(), bridge, new
            Door());
        corridore4.setEntrance(Directions.WEST.getValue(), storeRoom, new
             Door());
        storeRoom.setEntrance(Directions.NORTH.getValue(), null, new Door
            (true));
        bridge.setEntrance(Directions.SOUTH.getValue(), garden, new Door
            ());
        tmpDoor = new Door(true);
        tmpDoor.setUnlockItem(Items.SEAL);
        garden.setEntrance(Directions.WEST.getValue(), temple, tmpDoor);
        garden.setEntrance(Directions.EAST.getValue(), tavern, new Door()
            );
        temple.setEntrance(Directions.WEST.getValue(), teleport, new Door
            ());
        temple.setEntrance(Directions.SOUTH.getValue(), null, new Door(
            true));

        //Adds items and characters to the rooms
        entrance.addItem(Items.DOOR_SIGN);
        corridore1.addItem(Items.WARNING_SIGN);
        pit.addItem(Items.NOTE);
        pit.addItem(Items.SKELETON);
        diningRoom.addItem(Items.MENU);
        diningRoom.addItem(Items.CHAIRS);
        diningRoom.addItem(Items.TABLES);
        kitchen.addItem(Items.FOOD);
        kitchen.addItem(Items.TABLES);
        corridore2.addItem(Items.BENCH);
        Character beggar = new Beggar();
        characters.put(beggar.getName(), beggar);
        corridore2.addCharacter(beggar);
        storeRoom.addItem(Items.BOXES);
        storeRoom.addItem(Items.PRAYER_BEADS);
        garden.addItem(Items.BENCH);
        Character cat = new Cat();
        characters.put(cat.getName(), cat);
        garden.addCharacter(cat);
        Character templeGuard = new TempleGuard();
        characters.put(templeGuard.getName(), templeGuard);
        garden.addCharacter(templeGuard);
        temple.addItem(Items.STONE_OF_DELEN);
        teleport.addItem(Items.HOLY_STONE);
        tavern.addItem(Items.TABLES);
```

```java
        tavern.addItem(Items.CHAIRS);
        Character priest = new Priest();
        characters.put(priest.getName(), priest);
        tavern.addCharacter(priest);
    }

    //Creates the parser and adds all the possible commands.
    private void createParser() {
        parser = new Parser(this);
    }

    //Creates the player.
    private void createPlayer() {
        player = new Player(generateName());
    }

    /**
     * Generates a name to be used for characters.
     *
     * @return A string with 3-8 chars which can be used as a character
         name.
     */
    public static String generateName() {
        String vocals = "aeiouy";
        String consonants = "bcdfghjklmnpqrstvwxz";
        int nameLength = (int)(Math.random()*5+3);
        StringBuilder name = new StringBuilder(nameLength);
        //Creates a random name but with two rules:
        //1: a vocal should not be followed by a second vocal
        //2: there can't be more than 2 consonants in a row
        for (int i=0;i<nameLength;i++) {
            boolean vocal;
            if (i>0) {
                if (vocals.indexOf(name.charAt(i-1))!=-1) {
                    vocal = false;
                } else if(i>1&&consonants.indexOf(name.charAt(i-1))!=-1)
                    {
                    vocal = true;
                } else {
                    vocal = (Math.random()<0.3);
                }
            } else {
                vocal = (Math.random()<0.3);
            }
            if (vocal) {
                name.append(vocals.charAt((int)(Math.random()*vocals.
                    length())));
            } else {
                name.append(consonants.charAt((int)(Math.random()*
                    consonants.length())));
            }
        }
        name.setCharAt(0, java.lang.Character.toUpperCase(name.charAt(0))
            );

        String finalName = name.toString();
        //If the name is already used: generat a new
        if (usedNames.contains(finalName)) {
            return generateName();
        }
        usedNames.add(finalName);
        return finalName;
    }

    /**
     *  Main play routine. Loops until end of play.
     */
    public void play()
    {
        printWelcome();
        player.setCurrentRoom(getRoom(Rooms.ENTRANCE));
        while (notFinished) {
            parser.getCommand();
            moveCharacters();
        }
    }

    //Moves characters that are marked as walk randomly
    private void moveCharacters() {
        for (Character c : characters.values()) {
            if (c.isWalkingRandomly() && Math.random()<0.2) {
                Room cRoom = c.getCurrentRoom();
                String[] exits = cRoom.getExits();
                ArrayList<Room> possibleRooms = new ArrayList<Room>(exits
                    .length);
                for (int i=0;i<exits.length;i++) {
                    Room checkRoom = cRoom.getDoor(exits[i]).getExit(
                        cRoom);
                    if (checkRoom != null && checkRoom != getRoom(Rooms.
                        PIT)) {
                         possibleRooms.add(checkRoom);
                    }
                }
                if (possibleRooms.size()>0) {
                    Room newRoom = possibleRooms.get((int)(Math.random()*
                        possibleRooms.size()));
                    cRoom.removeCharacter(c);
                    newRoom.addCharacter(c);
                    if (cRoom == getPlayer().getCurrentRoom()) {
                        System.out.println(String.format("%s gick iväg.",
                            c.getName()));
                    } else if (newRoom == getPlayer().getCurrentRoom()) {
                        System.out.println(String.format("%s kom in i
                            rummet.", c.getName()));
                    }
                }
            }
        }
    }

    /*
     * Print out the opening message for the player.
     */
    private void printWelcome()
    {
        System.out.print("Du har blivit insläpad i borgen men det visade
            sig vara ett missförstånd. Vakten har gett sig iväg. Du vill
            bara komma ut igen men dörren är låst.");
        System.out.println(String.format(" Ditt namn är %s", getPlayer().
            getName()));
        System.out.println("Skriv 'hjälp' ifall du behöver hjälp eller '
            hjälp <kommando>' för att lista möjliga mål för ett kommando,
            t.ex: 'hjälp gå'.");
    }

    /**
     * Returns the player.
     *
     * @return The player.
     */
    public Player getPlayer() {
        return player;
    }

    /**
     * Gets the parser
     *
     * @return The parser.
     */
    public Parser getParser() {
        return parser;
    }

    /**
     * Gets a room from the map using the enum Rooms.
     *
     * @param name The enum linked to the room.
     * @return The room.
     */
    public Room getRoom(Rooms name) {
        return map.get(name);
    }

    /**
     * Ends the game.
     *
     * @param completed If <code>true</code> print a success message and
         if <code>false</code>
     * print a fail message.
     */
    public void endGame(boolean completed) {
        notFinished = false;
        if (completed) {
            System.out.println("Du klarade det, du kom ut ur borgen!");
        } else {
            System.out.println("Du har misslyckats!");
        }
    }

    /**
     * Saves the current game to a file so that it can be loaded later.
     *
     * @param f The file where the game will be saved.
     */

    public void saveGame(File f) {
        /*
         * Tries to save the game with an object output stream. Writes
             all the important classes
         * to the file so the information can be loaded later.
         */
        try {
            ObjectOutputStream out = new ObjectOutputStream(new
                FileOutputStream(f));
            out.writeObject(player);
            out.writeObject(map);
            out.writeObject(characters);
            out.flush();
            out.close();
        } catch (IOException e) {
            System.out.println("Misslyckades att spara filen med
                felmeddelandet: "+e.getMessage());
        }
    }

    @SuppressWarnings("unchecked")
    public void loadGame(File f) {
        /*
```

```
         * Tries to load the game from the file by using an object input stream.
         * It reads one object at a time and tries to cast it to the correct class.
         * If any of these casts or the reading would fail the game could not be loaded.
         */
        try {
            ObjectInputStream in = new ObjectInputStream(new FileInputStream(f));
            Player tmpPlayer = (Player)in.readObject();
            EnumMap<Rooms, Room> tmpMap = (EnumMap<Rooms, Room>)in.readObject();
            HashMap<String, Character> tmpCharacter = (HashMap<String, Character>)in.readObject();
            player = tmpPlayer;
            map = tmpMap;
            characters = tmpCharacter;
            in.close();
        } catch (IOException e) { //Problem with the stream
            System.out.println("Misslyckades att ladda filen med felmeddelandet: "+e.getMessage());
        } catch (ClassNotFoundException e) { //Problem reading a class from the stream
            System.out.println("Kan inte ladda sparat spel, troligen för att det är av en gammal version av spelet.");
        } catch (ClassCastException e) { //The wrong class was read from the stream.
            System.out.println("Kan inte ladda sparat spel, troligen för att det är av en gammal version av spelet.");
        }
    }
}
```

Item

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Item.java

```
package org.x2d.zuul;
import java.io.*;
/**
 * Abstract class representing an item. Methods should be overridden
 * if the new class should do scriptet stuff. As an example:
<pre>
new Item("some name", "a description of the item") {
    &#0064;Override
    public boolean isUsable() {
        return true;
    }
    &#0064;Override
    public void use(Game g) {
        //Something that should happen when this item is used.
    }
};
</pre>
 */
public abstract class Item implements Serializable
{
    private String name;
    private String description;
    private String text;
    private boolean isTakable;

    /**
     * Constructor for objects of class Item
     * @param name The name of the item.
     * @param description A short description of the item.
     */
    public Item(String name, String description) {
        this(name, description, null, false);
    }

    /**
     * Constructor for objects of class Item
     * @param name The name of the item.
     * @param description A short description of the item.
     * @param text A text that should be used when the item is read.
     * @param isTakable Should be <code>true</code> if the item could be
         put into the backpack else <code>false</code>
     */
    public Item(String name, String description, String text, boolean
        isTakable) {
        this.name = name;
        this.description = description;
        this.text = text;
        this.isTakable = isTakable;
    }

    /**
     * Checks if it's possible to read from this item
     * @return <code>true</code> if there is text on the item else <code>
         false</code>
     */
    public boolean isReadable() {
        return (getText()!=null);
    }

    /**
     * Gets this item's description
     *
     * @return The description.
     */
    public String getDescription() {
        return description;
    }

    /**
     * Gets this item's name
     *
     * @return The name.
     */
    public String getName() {
        return name;
    }

    /**
     * Checks if it's possible to pickup this item
     * @return <code>true</code> if there is text on the item else <code>
         false</code>
     */
    public boolean isTakable() {
        return isTakable;
    }

    /**
     * Gets the text from this item
     * @return The text or null if there is none.
     */
    public String getText() {
        return text;
    }

```

```java
    /**
     * This method should be called when the item is used.
     *
     * @param g The game which this item should interact with
     */
    public abstract void use(Game g);

    /**
     * Checks if it's possible to use this item. This should return true if there is an use method.
     * @return <code>true</code> if it is possible to use this item else <code>false</code>
     */
    public abstract boolean isUsable();
}
```

Items

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Items.java

```java
package org.x2d.zuul;
import java.util.*;
import java.io.*;
/**
 * Enum that has all the items used in the game. It has one single public method to get an item from it.
 *
 */
public enum Items implements Serializable {

    STONE_OF_DELEN(new SimpleItem("sten av delen", "En skimrande sten med märkliga tecken.", "G&a", true)),
    STORE_ROOM_KEY(new SimpleItem("lagernyckel", "En vanlig nyckel.", null, true)),
    NOTE(new SimpleItem("lapp", "En gammal utsliten lapp.", "Meningen med liver är 42!", true)),
    SKELETON(new SimpleItem("skelett", "Ett skelett som ser ut att ha legat här i evigheter.")),
    FOOD(new SimpleItem("mat", "Ett bröd och lite ost.", null, true)),
    PRAYER_BEADS(new SimpleItem("bönband", "En rad med pärlor på ett snöre.", null, true)),
    WARNING_SIGN(new SimpleItem("varningsskylt", "En skylt med en döskalle på.",
        "Farligt område! Beträds på egen risk!", false)),
    DOOR_SIGN(new SimpleItem("dörrskylt", "En skylt som sitter brevid dörren.",
        "Dörren går endast att öppna med hjälp magisk sten.", false)),
    EATEN_SANDWICH(new SimpleItem("uppäten macka", "En uppäten macka. Inte speciellt mycket mat på den.", null, true)),
    MENU(new SimpleItem("meny", "En lista med mat.", "Stekt kött 2öre\nBröd 1öre\nOst 2öre", false)),
    CHAIRS(new SimpleItem("stolar", "Ett antal stolar.")),
    TABLES(new SimpleItem("bord", "Ett antal bord.")),
    BENCH(new SimpleItem("bänk", "En bänk.")),
    BOXES(new SimpleItem("lådor", "Massor av lådor som står staplade längs väggarna.")),
    SEAL(new SimpleItem("sigill", "Ett brev med sigill från prästen.",
        "Jag intygar att personen som innehar detta brev får komma in i templet.", true)),
    HOLY_STONE(new Item("helig sten", "En stor sten med konstiga tecken på.") {
        public boolean isUsable() {
            return true;
        }
        public void use(Game g) {
            System.out.println(
                "Allting runtomkring dig blir suddigt och plötsligt ser
                    du att du befinner dig i ett helt annat rum.");
            g.getPlayer().setCurrentRoom(g.getRoom(Game.Rooms.ENTRANCE));
        }
    });

    private static HashMap<String, Items> itemMap = new HashMap<String,
        Items>();
    static {
        for (Items item : Items.values()) {
             itemMap.put(item.getItem().getName(), item);
        }
    }
    private Item item;
    private Items(Item item) {
        this.item = item;
    }

    /**
     * Gets an item by it's name.
     *
     * @param itemName The name.
     */
    public static Items getItem(String itemName) {
        return itemMap.get(itemName);
    }

    /**
     * Gets the item this enum is representing.
     * @return The item.
     */
    public Item getItem() {
        return item;
    }
}
```

Parser

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Parser.java

```
package org.x2d.zuul;
import java.util.*;
/**
 * This class is part of the "World of Zuul" application.
 * "World of Zuul" is a very simple, text based adventure game.
 *
 * This parser reads user input and tries to interpret it as an "
     Adventure"
 * command. Every time it is called it reads a line from the terminal and
 * tries to interpret the line as a two word command. It returns the
     command
 * as an object of class Command.
 *
 * The parser has a set of known command words. It checks user input
     against
 * the known commands, and if the input is not one of the known commands,
     it
 * returns a command object that is marked as an unknown command.
 *
 * @author  Michael Kolling and David J. Barnes
 * @version 2008.03.30
 */
public class Parser
{
    private static final HashMap<String, CommandWord> validCommands =
        new HashMap<String, CommandWord>(10);
    private Game game;
    private static Scanner reader = new Scanner(System.in);         //
        source of command input

    /**
     * Create a parser to read from the terminal window.
     */
    public Parser(Game game)
    {
        this.game = game;
        addCommandWord(new CommandGo("gå"));
        addCommandWord(new CommandUnlock("låsupp"));
        addCommandWord(new CommandWord("sluta") {
                public void executeCommand(Game game, String target) {
                    game.endGame(false);
                }
            });
        addCommandWord(new CommandHelp("hjälp"));
        addCommandWord(new CommandUse("använd"));
        addCommandWord(new CommandTake("ta"));
        addCommandWord(new CommandTalk("prata"));
        addCommandWord(new CommandRead("läs"));
        addCommandWord(new CommandSave("spara"));
        addCommandWord(new CommandLoad("ladda"));
        addCommandWord(new CommandList("lista"));
    }

    /**
     * Reads one line parses it for and commands. If a command is found
         then
     * it's executed.
     */
    public void getCommand()
    {
        String inputLine;   // will hold the full input line
        String command = null;
        String target = null;

        inputLine = reader.nextLine().trim();
        System.out.println("> "+inputLine);
        int spacePos = inputLine.indexOf(' ');
        if (spacePos == -1) {
            command = inputLine;
        } else {
            command = inputLine.substring(0, spacePos);
            target = inputLine.substring(spacePos+1);
        }
        if (!isCommand(command)) {
            System.out.println("Okänd kommando: "+command);
            getCommand("hjälp").executeCommand(getGame(), null);
            return;
        }
        CommandWord cw = getCommand(command);
        if (cw.mustHaveTarget()&&target==null) {
            System.out.println("Det här kommandot kräver ett mål, vad
                vill du använda det på?");
            getCommand("hjälp").executeCommand(getGame(), command);
            return;
        }
        cw.executeCommand(getGame(), target);
    }

    /**
     * Gets the game currently using this parser.
     *
     * @return The game.
```

```java
     */
    public Game getGame() {
        return game;
    }

    /**
     * Sets which game that is currently using this parser.
     *
     * @param game The game.
     */
    public void setGame(Game game) {
        this.game = game;
    }

    /**
     * Returns a list of all possible targets for a command.
     *
     * @param command The command's name
     * @return The list of possible targets or <code>null</code> if it's
         not a valid command
     */
    public String[] showTargets(String command) {
        if (isCommand(command)) {
            return getCommand(command).getTargets(getGame());
        }
        return null;
    }

    /**
     * Adds a command word to the list of commands.
     *
     * @param cw A CommandWord which should be added to this list of
         commands.
     */
    public void addCommandWord(CommandWord cw) {
        validCommands.put(cw.getCommand(), cw);
    }

    /**
     * Check whether a given String is a valid command word.
     *
     * @return <code>true</code> if it is, <code>false</code> if it isn't
         .
     */
    public boolean isCommand(String aString)
    {
        return (validCommands.get(aString)!=null);
    }

    /**
     * Gets a collection of the command words.
     *
     * @return A collection of all the command words.
     */
    public Collection<CommandWord> getCommandWords() {
        return validCommands.values();
    }

    /**
     * Gets a single command word from the list of command words.
     *
     * @return The command word.
     */
    public CommandWord getCommand(String command) {
        return validCommands.get(command);
    }
```

```

}
```

Player

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Player.java

```java
package org.x2d.zuul;
import java.util.*;
import java.io.*;
/**
 * Class representing the player. The player has a backpack
 * which stores item.
 */
public class Player implements Serializable {
    private String name;
    private HashSet<Items> items = new HashSet<Items>(10);
    private Room currentRoom;

    /**
     * Constructor for objects of class Player
     * @param name The name of the player.
     */
    public Player(String name) {
        this.name = name;
    }

    /**
     * Gets the name of the player
     *
     * @return The player's name
     */
    public String getName() {
        return name;
    }

    /**
     * Sets the player's name
     *
     * @param name The player's new name.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Adds an item to the players backpack
     *
     * @param item The item to add.
     */
    public void addItem(Items item) {
        items.add(item);
        System.out.println(String.format("Du tar imot %s och lägger i din
             ryggsäck.", item.getItem().getName()));
    }

    /**
     * Removes an item from the players backpack
     *
     * @param item The item to remove.
     */
    public void removeItem(Items item) {
        items.remove(item);
        System.out.println(String.format("Du plockar upp %s ur din
            räcksäck och använder.", item.getItem().getName()));
    }

    /**
     * Gets an item from the players backpack
     *
     * @param item The item
     * @return <code>true</code> if found else <code>false</code>
     */
    public boolean hasItem(Items item) {
        return items.contains(item);
    }

    /**
     * Gets a collection of all the items in the player's backpack
     *
     * @return A collection of all the items
     */
    public Collection<Items> getItems() {
        return items;
    }

    /**
     * Gets the room current which the player currently is in.
     *
     * @return The current room.
     */
    public Room getCurrentRoom() {
        return currentRoom;
    }

    /**
     * Sets the room current which the player currently is in.
     *
     * @param cRoom The the room which the player should now be in.
     */
    public void setCurrentRoom(Room cRoom) {
        currentRoom = cRoom;
        System.out.println();
        System.out.println(currentRoom.getLongDescription());
    }
    /**
     * Try to go to one direction. If there is an exit, enter the new
     * room, otherwise print an error message.
     */
    public void goRoom(String direction)
    {
        // Try to leave current room.
        Door door = getCurrentRoom().getDoor(direction);
        if (door.isLocked()) {
            Items unlockItem = door.getUnlockItem();
            if (unlockItem!=null) {
                System.out.println(String.format("Dörren är låst och du
                    behöver: %s för att låsa upp dörren..", unlockItem.
                    getItem().getName()));
            } else {
                System.out.println("Dörren är låst.");
            }
            return;
        }
        Room nextRoom = door.getExit(getCurrentRoom());
        setCurrentRoom(nextRoom);

    }
}
```

Priest

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Priest.java

```java
package org.x2d.zuul;
/**
 * The priest NPC that has lost his prayer beads
 * and when the player gives them to the priest
 * he gives player a seal which will let him get into
 * the temple.
 */
public class Priest extends Character {
    private Item seal;
    boolean gotPrayerBeads = false;
    private int tCounter1 = 0;
    private int tCounter2 = 0;
    public Priest() {
        super(Game.generateName(), "Argh!");
    }

    @Override public void talk(Game g, Player p) {
        //Replies when the player has given the Priest the prayer beads
        if (gotPrayerBeads) {
            String[] texts = {
                "Gud är stor!",
                "Jag hoppas att du får all välgång!",
            };
            System.out.println(texts[tCounter1]);
            tCounter1 = (tCounter1 + 1) % texts.length;
            return;
        }
        //Some replies to give the player a clue of what the priest want
            for an item
        if (!p.hasItem(Items.PRAYER_BEADS)) {
            String[] texts = {
                "Jag förstår inte vart mitt böneband har tagit vägen!",
                "Du kanske kan hjälpa mig att hitta den?",
                "Jag vet att jag hade den när jag var och åt.",
                "Jag var inne i lagerrummet sen."
            };
            System.out.println(texts[tCounter1]);
            tCounter1 = (tCounter1 + 1) % texts.length;
        //If the player has the prayer beads give it to the priest and
            the player gets
        //a seal which will help him get into the temple.
        } else {
            System.out.println("Ah du hittade den!");
            p.removeItem(Items.PRAYER_BEADS);
            System.out.println("Aha, du vill komma ut ur borgen.");
            System.out.println("Jag tror du kan hitta det du letar efter
                i templet.");
            System.out.println("Här får du ett mitt sigill som du kan
                visa för vakten så att du kommer in.");
            p.addItem(Items.SEAL);
            gotPrayerBeads = true;
        }
    }
}
```

Room

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/Room.java

```java
package org.x2d.zuul;
import java.util.Set;
import java.util.HashMap;
import java.util.Iterator;
import java.util.*;
import java.io.*;
```

```java
/**
 * Class Room - a room in an adventure game.
 *
 * A "Room" represents one location in the scenery of the game.  It is
 * connected to other rooms via doors.  For each existing exit, the room
 * stores a reference to the door.
 *
 * The room also has items and doors which the player can interact with.
 */
public class Room implements Serializable {
    private String description;
    private HashSet<Items> items = new HashSet<Items>(2);
    private HashMap<String, Door> exits;        // stores exits of this room.
    private HashMap<String, Character> characters;

    /**
     * Create a room described "description". Initially, it has
     * no exits. "description" is something like "a kitchen" or
     * "an open court yard".
     * @param description The room's description.
     */
    public Room(String description) {
        this.description = description;
        exits = new HashMap<String, Door>();
        characters = new HashMap<String, Character>();
    }

    /**
     * Define an exit from this room. If neighbor is not null then
     * it will add the door to that room too.
     * @param direction The direction of the exit.
     * @param neighbor  The room to which the exit leads.
     */
    public void setEntrance(String direction, Room neighbor, Door door) {
        ArrayList<String> directions = new ArrayList<String>(4);
        directions.add(Game.Directions.NORTH.getValue());
        directions.add(Game.Directions.EAST.getValue());
        directions.add(Game.Directions.SOUTH.getValue());
        directions.add(Game.Directions.WEST.getValue());
        door.setRoom1(this);
        exits.put(direction, door);
        if (neighbor!=null) {
            //Gets the other side's exit direction, so south return north and so on
            String oppositeDirection = directions.get((directions.indexOf(direction)+2)%4);
            neighbor.setExit(oppositeDirection, door);
        }
    }

    /**
     * Define an exit from this room. The difference from setEntrance is that this
     * only sets a door to an direction and does not care where the door goes.
     * @param direction The direction of the exit.
     * @param door The door that should be in that direction.
     */
    public void setExit(String direction, Door door) {
        if (door!=null) {
            door.setRoom2(this);
            exits.put(direction, door);
        } else {
            exits.remove(direction);
        }

    }

    /**
     * @return The short description of the room
     * (the one that was defined in the constructor).
     */
    public String getShortDescription() {
        return description;
    }

    /**
     * Return a description of the room
     * @return A long description of this room
     */
    public String getLongDescription() {
        StringBuilder tmpString = new StringBuilder();
        tmpString.append(description).append("\n");
        Collection<Items> items = getItems();
        if (items.size() != 0) {
            tmpString.append(String.format("Det finns %d föremål i rummet
                :\n", items.size()));
            for (Items item : getItems()) {
                Item itemObject = item.getItem();
                tmpString.append(" - ").append(itemObject.getName()).
                    append(": ");
                tmpString.append(itemObject.getDescription()).append("\n"
                    );
            }
        }
        Collection<Character> characters = getCharacters();
        if (characters.size() != 0) {
            tmpString.append(String.format("There finns %d person%s/djur
                i rummet:\n", characters.size(), (characters.size()==1)?""
                :"er"));
            for (Character character : characters) {
                tmpString.append(" - ").append(character.getName());
                if (character.isFirstTime()) {
                    tmpString.append(" säger: ").append(character.
                        getFirstTimeText());
                    character.setFirstTime(false);
                }
                tmpString.append("\n");
            }
        }
        tmpString.append(getExitString());
        return tmpString.toString();
    }

    /**
     * Return a string describing the room's exits, for example
     * "Exits: north west".
     * @return Details of the room's exits.
     */
    private String getExitString() {
        StringBuilder returnString = new StringBuilder("Utgångar:");
        Set<String> keys = exits.keySet();
        if (keys.size()!=0) {
            for(String exit : keys) {
                returnString.append(" ").append(exit);
            }
        } else {
            returnString.append(" inga");
        }
        return returnString.toString();
    }
    /**
     * Gets a door from this room by direction.
     *
     * @param direction The direction.
     * @return the door or null if it doesn’t exist
     */
    public Door getDoor(String direction) {
        return exits.get(direction);
    }

    /**
     * Adds an item to this room.
     * @param item The item
     */
    public void addItem(Items item) {
        items.add(item);
    }

    /**
     * Removes an item from this room.
     * @param item The item
     */
    public void removeItem(Items item) {
        items.remove(item);
    }

    /**
     * Gets an item by its name
     * @return The item.
     */
    public boolean hasItem(Items item) {
        return items.contains(item);
    }

    /**
     * Gets an collection with all the items in this room
     *
     * @return All the items in this room
     */
    public Collection<Items> getItems() {
        return items;
    }

    /**
     * Gets all the directions where there are doors.
     *
     * @return The directions.
     */
    public String[] getExits() {
        return exits.keySet().toArray(new String[0]);
    }

    /**
     * Adds a character to this room.
     *
     * @param c The character
     */
    public void addCharacter(Character c) {
        characters.put(c.getName(), c);
        c.setCurrentRoom(this);
    }

    /**
     * Removes a character from this room.
     *
     * @param c The character
     */
    public void removeCharacter(Character c) {
        characters.remove(c.getName());
    }

    /**
     * Gets a character by his/her name
     * @return The item.
     */
    public Character getCharacter(String name) {
        return characters.get(java.lang.Character.toUpperCase(name.charAt
            (0))+name.substring(1));
    }

    /**
     * Gets an collection with all the characters in this room
     *
     * @return All the characters in this room
     */
    public Collection<Character> getCharacters() {
        return characters.values();
    }
}
```

SaveGameFilter

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/SaveGameFilter.java

```
package org.x2d.zuul;
import java.io.File;
import java.io.FilenameFilter;
/**
 * Class used as a filter for zuul save game files.
 */
public class SaveGameFilter implements FilenameFilter {
    /**
     * ".zul"
     */
    public static final String SAVE_GAME_EXTENSION = ".zul";
    /**
     * Creates a new filter that only matches files ending with ".zul".
     */
    public SaveGameFilter() {
    }
    /**
     * Accepts only .zul files
     *
     * @return <code>true</code> if the file ends with .zul else <code>
         false</code>
     */
    public boolean accept(File directory, String filename) {
        return filename.endsWith(SAVE_GAME_EXTENSION);
    }
}
```

SimpleItem

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/SimpleItem.java

```
package org.x2d.zuul;
/**
 * Version of Item does not require an use method.
 */
public class SimpleItem extends Item {
    /**
     * Constructor for objects of class Item
     * @param name The name of the item.
     * @param description The item's description
     */
    public SimpleItem(String name, String description) {
        super(name, description);
    }

    /**
     * Constructor for objects of class Item
     * @param name The name of the item.
     * @param description The item's description
     * @param text If there should be some text on the item that can
     * be read. If set to <code>null</code> there is nothing to read on
         the item.
     * @param isTakable Should be <code>true</code> if it is possible to
         take the
     * item else it should be <code>false</code>
     */
    public SimpleItem(String name, String description, String text,
        boolean isTakable) {
        super(name, description, text, isTakable);
    }

    @Override
    public void use(Game g) {
        //Can't be used
    }

    @Override
    public boolean isUsable() {
        return false;
    }

}
```

TempleGuard

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/zuul/TempleGuard.java

```
package org.x2d.zuul;
/**
 * A guard that will not the let the player enter the temple
 * till he sees a paper with a seal from the priest.
 */
public class TempleGuard extends Character {
    public TempleGuard() {
        super(Game.generateName(), "");
        setFirstTime(false);
    }

    public void talk(Game g, Player p) {
        //If the door is open.
        if (!g.getRoom(Game.Rooms.GARDEN).getDoor(Game.Directions.WEST.
            getValue()).isLocked()) {
            String[] texts = {
                "Det är bara att gå in i templet.",
            };
            System.out.println(texts[0]);
            return;
        }
        //If the door is locked and the player doesn't have the seal
        if (!p.hasItem(Items.SEAL)) {
            String[] texts = {
                "Här kommer du inte förbi!",
                "Jag släpper inte in något utan tillstånd."
            };
            System.out.println(texts[(int)(Math.random()*texts.length)]);
        //If the door is locked and the player has the seal
        //the guard unlocks the
        } else {
            System.out.println("Här kommer du...");
            System.out.println("Vad är det där för papper?");
            System.out.println("Jaha, ja då kommer du in, jag öppnar dörren.");
            p.removeItem(Items.SEAL);
            g.getRoom(Game.Rooms.GARDEN).getDoor("väster").setLocked(false);
            System.out.println("Den västra dörren är nu upplåst.");
        }
    }
}
```

Console

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/console/Console.java

```
package org.x2d.console;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;
import java.io.*;
/**
 * An textarea in a JScrollPane that displays all the text which is sent
 * to System.out
 */
public class Console extends JScrollPane implements Runnable {
    private final int maxLines;
    private PipedOutputStream pout;
    private final PipedInputStream pin = new PipedInputStream();
    private BufferedReader in;
    private Thread t;
    short[] newLinePos;
    private int newLineCounterPos = 0;
    private boolean linesFull = false;
    private JTextArea textArea = new JTextArea();
    boolean scrollNext = false;
    boolean firstLine = true;

    /**
     * Creates a new console with a maximum number of lines. When
     * created it will replace Syste.out so all it's output is
     * redirected to this console. If a second console is created the old
     * console will stop working.
     *
     * @param maxLines the maximum number of lines this console can show.
     */
    public Console(int maxLines) {
        super(VERTICAL_SCROLLBAR_ALWAYS, HORIZONTAL_SCROLLBAR_NEVER);
        if (maxLines<1) {
            throw new IllegalArgumentException("The number of lines must be greater than 0.");
        }
        getViewport().setView(textArea);
        this.maxLines = maxLines;
        newLinePos = new short[maxLines];
        textArea.setEditable(false);
        textArea.setLineWrap(true);
        textArea.setWrapStyleWord(true);
        final JScrollBar scroll = getVerticalScrollBar();
        scroll.addAdjustmentListener(new AdjustmentListener(){
            public void adjustmentValueChanged(AdjustmentEvent e){
            //Scrolls to the bottom of the text area if needed.
                if (scrollNext) {
                    scroll.setValue(scroll.getMaximum()-scroll.getVisibleAmount());
                    scrollNext=false;
                }
            }});
        /*
         * Redirects system.out
         */
        try {
            pout = new PipedOutputStream(pin);
            System.setOut(new PrintStream(pout,true));
            t = new Thread(this);
            t.setDaemon(true);
            t.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Used internally to read the System.out buffer for new text.
     */
    public void run() {
        try {
            while (true) {
                try {
                    t.sleep(100);
                }catch(InterruptedException ie) {}
                if (pin.available()!=0) {
                    addLine(readLine(pin));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * Adds a line to this console. Synchronized makes sure that
     * only one thread can use this object at any given time.
     *
     * @param line Appends line to this console.
     */
    public synchronized void addLine(String line) {
        /*
         * Makes sure that the textarea shows at maximum <i>maxLine</i> lines
         * of text and that the scroll stays at the bottom if
         * it was at the bottom before the new line was added.
         */

        JScrollBar scroll = getVerticalScrollBar();
        int max = scroll.getMaximum();
        int value = scroll.getValue();
        int visible = scroll.getVisibleAmount();
        if (max == value+visible) {
            scrollNext = true;
        }
        if (newLineCounterPos==maxLines-1) {
            linesFull = true;
        }
        newLineCounterPos = (newLineCounterPos+1)%maxLines;
        if (linesFull) {
            textArea.replaceRange(null, 0, (int)newLinePos[
                newLineCounterPos]);
        }
        int newPos=line.length();
        if (firstLine) {
            firstLine=false;
            newPos += 1;
        }
        newLinePos[newLineCounterPos] = (short)newPos;
        textArea.append(line);

    }

    /*
     * Reads one line from the pipe and returns it. Synchronized makes
         sure that
     * only one thread can use this object at any given time.
     */
    private synchronized String readLine(PipedInputStream in) throws
        IOException {
        StringBuilder input;
        if (firstLine) {
            input = new StringBuilder();
        } else {
            input = new StringBuilder("\n");
        }
        int end=0;
        //Reads bytes from the stream till it finds a '\n'
        do {
            int available=in.available();
            if (available==0) break;
            byte b[]=new byte[available];
            in.read(b);
            input.append(new String(b,0,b.length));
            end = input.length()-1;
        } while( input.charAt(end)!='\n' );
        if (input.charAt(end-1)=='\r') {
            end--;
        }
        return input.substring(0, end);
    }
}
```

ConsoleGUI

/home/axel/Projekt/Skola/Inda/zuul/org/x2d/console/ConsoleGUI.java

```
package org.x2d.console;
import javax.swing.*;
import java.util.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
/**
 * Displays a frame with a textarea to show output from System.out and
 * a textfield to send commands to System.in
 * This can replace the terminal except for the error stream
 * (System.err) which still will be printed in the terminal.
 */
public class ConsoleGUI implements KeyListener {
    private static final int HISTORY_LENGTH = 10;
    private JFrame frame;
    private static final int MAIN_MENU = 0;
    private JMenuBar menuBar;
    private JMenu[] menus = {
            new JMenu("Main")
    };
    private Console con;
    private JButton buttonSend;
    private JTextField textFieldSend;
    private JPanel inputPanel;
    private PipedOutputStream pout;
    private PipedInputStream pin;
    private PrintStream out;
    private static int historyLength = 0;
    private int historyCounter = 0;
    private LinkedList<String> commandHistory = new LinkedList<String>();

    /**
     * Constructor - Setups a frame and redirects System.out and System.
         in
     * so this class displays the output and can send commands.
     * Sets the history length to 10.
     */
    public ConsoleGUI() {
        this(HISTORY_LENGTH);
    }

    /**
     * Constructor - Setups a frame and redirects System.out and System.
         in
     * So this class displays the output and can send commands.
     *
     * @param historyLength The number of commands saved as a history.
     */
    public ConsoleGUI(int historyLength) {
        if (historyLength<0) {
            throw new IllegalArgumentException("The history length must
                be greater than or equal to 0.");
        }
        frame = new JFrame("Terminal");
        frame.setBounds(100, 100, 600, 400);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        //Creates menus
        menuBar = new JMenuBar();
        for (JMenu m : menus) {
            menuBar.add(m);
        }
        menus[MAIN_MENU].add(new AbstractAction("Quit") {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.setJMenuBar(menuBar);
        //Creates a console
        con = new Console(10);
        frame.getContentPane().add(con, BorderLayout.CENTER);
        //Creates an input field
        inputPanel = new JPanel(new BorderLayout());
        textFieldSend = new JTextField();
        textFieldSend.addKeyListener(this);
        AbstractAction send = new AbstractAction("Send") {
            public void actionPerformed(ActionEvent e) {
                sendTextFromField();
            }
        };
        textFieldSend.addActionListener(send);
        buttonSend = new JButton(send);
        inputPanel.add(textFieldSend, BorderLayout.CENTER);
        inputPanel.add(buttonSend, BorderLayout.EAST);

        frame.getContentPane().add(inputPanel, BorderLayout.SOUTH);
        //Creates a pipe so the textfield writes to System.in
        try {
            pin = new PipedInputStream();
            pout = new PipedOutputStream(pin);
            out = new PrintStream(pout);
            System.setIn(pin);
        } catch (Exception e) {
            e.printStackTrace();
        }
        frame.setVisible(true);
        textFieldSend.requestFocusInWindow();
    }

    //Sends the text from the text field to Systm.in
    private void sendTextFromField() {
        out.println(textFieldSend.getText());
        out.flush();
        commandHistory.add(textFieldSend.getText());
        int historySize = commandHistory.size();
        if (historySize>HISTORY_LENGTH) {
            commandHistory.removeFirst();
            historySize--;
        }
        historyCounter = historySize;
        textFieldSend.setText(null);
        textFieldSend.requestFocus();

    }
    //Checks for UP/DOWN keys to browse the command history
    public void keyPressed(KeyEvent keyEvent) {
        switch (keyEvent.getKeyCode()) {
            case KeyEvent.VK_UP:
                historyCounter-=1;
                break;
            case KeyEvent.VK_DOWN:
                historyCounter+=1;
                break;
            default:
                return;
        }
        int historySize = commandHistory.size();
        if (historySize==0) {
            return;
        } else if (historyCounter<0) {
            historyCounter = historySize-1;
        } else {
            historyCounter = historyCounter%historySize;
        }
        textFieldSend.setText(commandHistory.get(historyCounter));
    }

    //Not used but must be created in the interface KeyListener
    public void keyReleased(KeyEvent keyEvent) {
    }

    public void keyTyped(KeyEvent keyEvent) {
    }
}
```