MIAMI

CSC 431

CharacterSmith3.5e

System Architecture Specification (SAS)

Team #12

Jefferson Boyd	Project Leader
Nathan Fitzpatrick	Front-End Developer
Alex Madarese	Back-End Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	3/29/2022	Jefferson Boyd Nathan Fitzpatrick Alex MAadarese	

Table of Contents

Table of Figures

1. System Analysis

1.2 System Overview

The focus of our system revolves around the entire idea of character creation. This character creator requires user input to design their desired D&D custom character. Every feature in the character creator must work in order for the system to function properly.

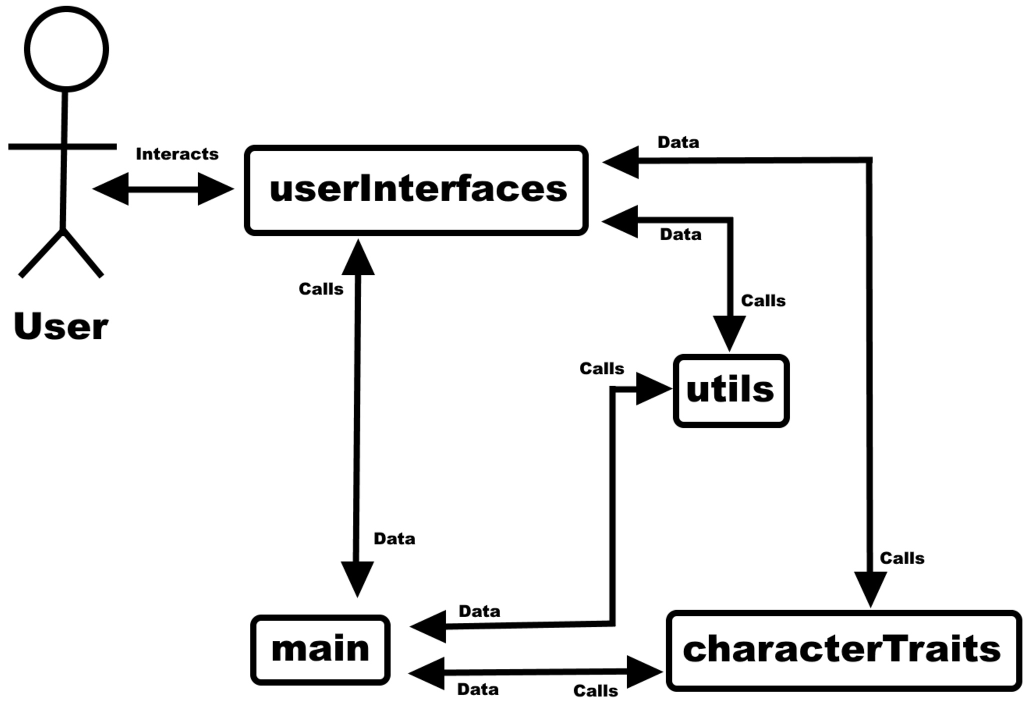
Some important features include:

- User Interface
- Main
- Character Traits
- Utils

The system consists of a user interface that can allow interaction between the user and the application. The user interface is also connected between every feature. Our main method stores all the static data that was initialized by the start of the application and also stores the character traits and utils chosen by the user. Character traits and utils are static and restrictive once chosen but can be reset at any point by the user.

The system will function locally and offline, not requiring internet connection to work once downloaded onto the device, so the user should have no issues using the character creator. The interface will function from a java applet that will allow the user to develop the character in a way they choose, show what was chosen, and then export the final product. However since our application is a creation tool, our style of architecture will be that of a Pipe and Filter.

1.3 System Diagram



1.4 Actor Identification

Actors that are interacting with the system should only be the users currently using the java applet. Those users are individuals that can start up the applet to create a character. There will be no need for a login to use our application or restrict others.

1.5 Design Rationale

1.5.1 Architectural Style

The architectural style this application will utilize is the Pipe and Filter style. The entire process should be obtaining as many options the user has inputted and making sure all the characteristics of the character are accepted and properly meets requirements based on their chosen attributes.

1.5.2 Design Pattern(s)

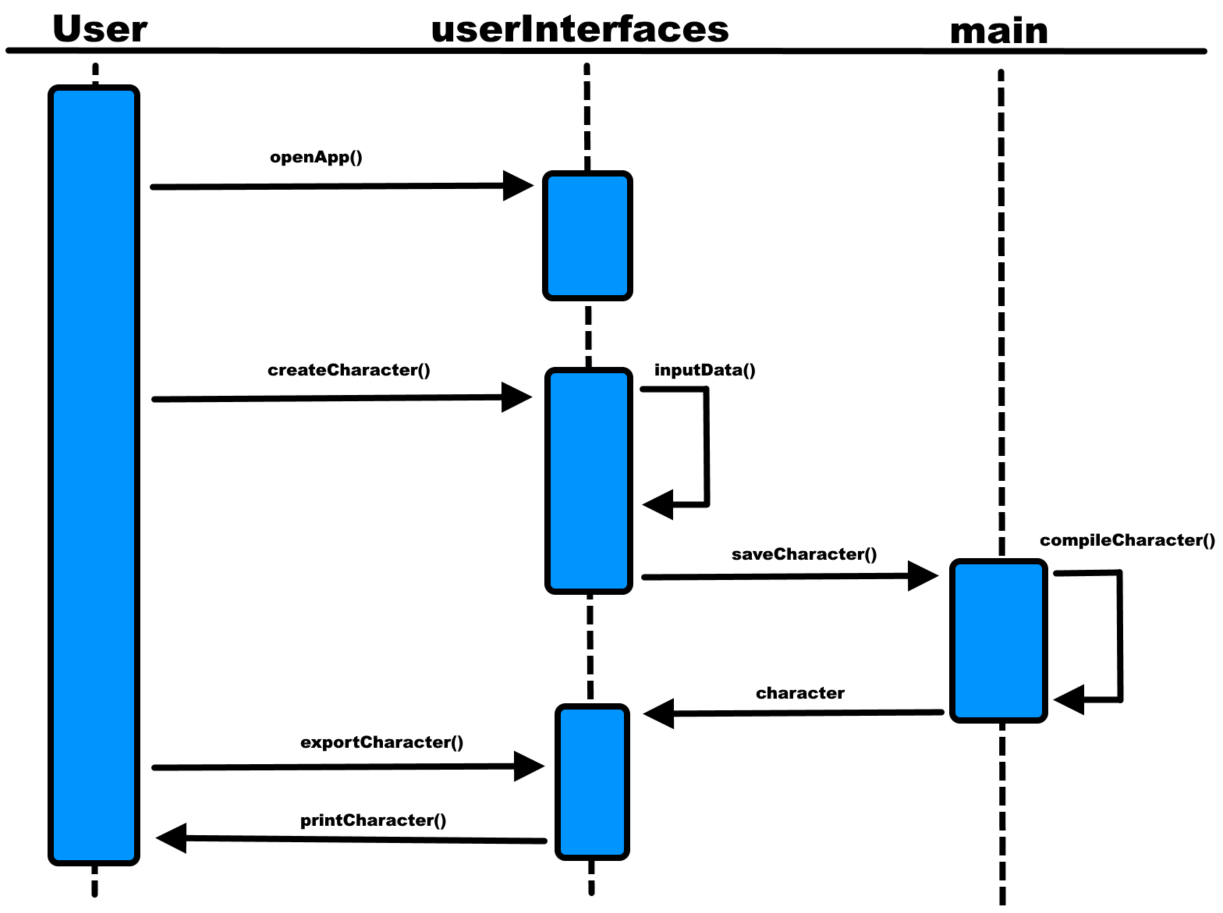
The design patterns we plan on using are the factory method and façade method. We are using a factory method, as we will be creating multiple similar classes using the same interfaces. Additionally, we plan on using objects in order to store instances of data, and as such will be using classes to make multiple different instances. We will be using a façade design pattern in order to present the user with a simple, and straightforward interface despite having several complex subsystems within the software.

1.5.3 Framework

Our application will utilize a form of a java applet, primarily using Java for frontend and backend development with Jswing library. Java swing allows for a cleaner graphical user interface for our program and a more convenient choice for Java and our desktop style application.

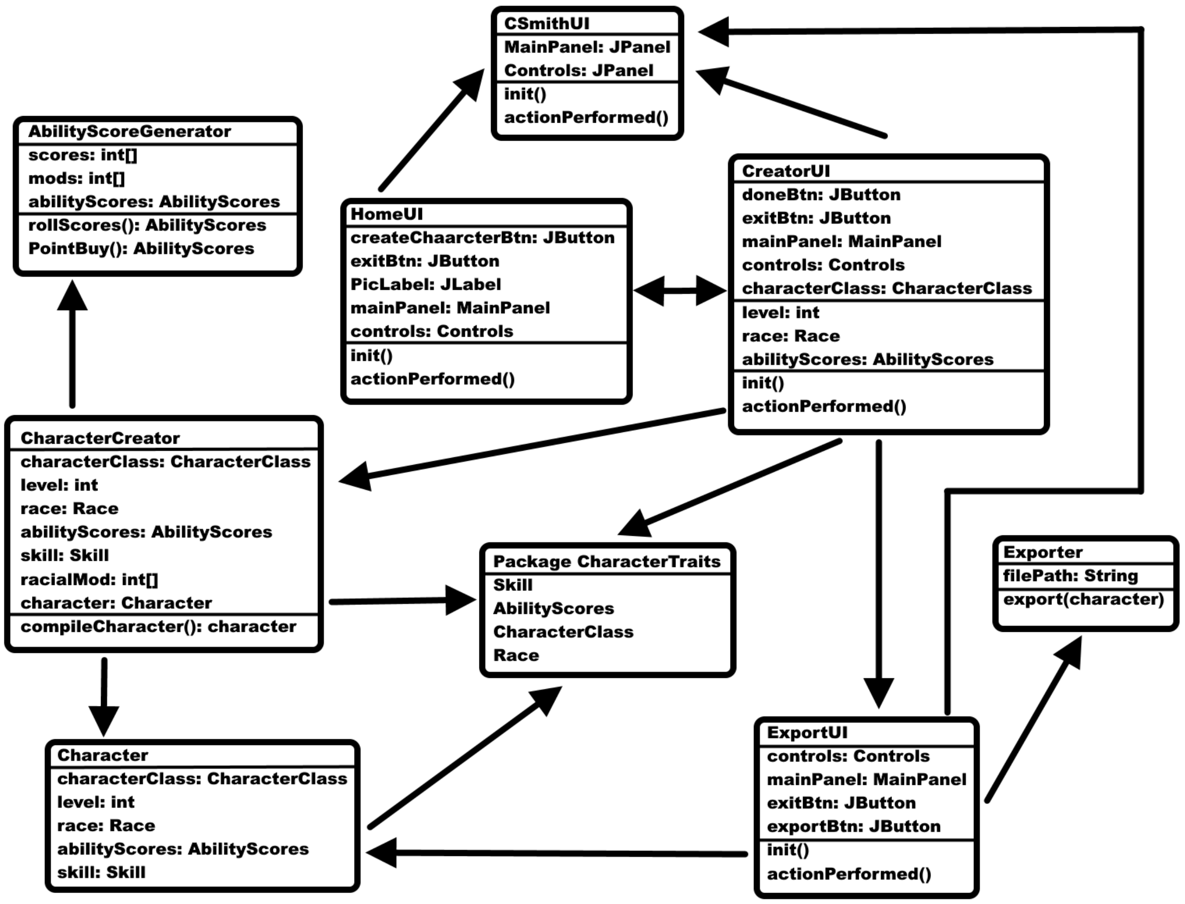
2. Functional Design

2.1 Character Creation

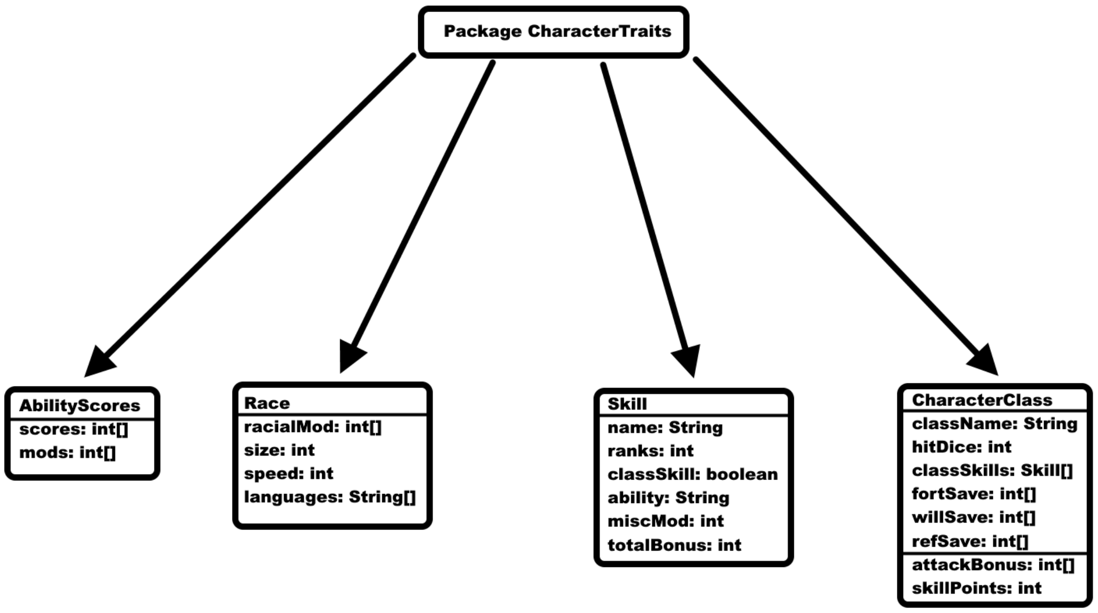


- When a user starts the program, the user interface calls openApp()
- The user interface then gives them options (create, export, print, or exit)
- Choosing to create a character allows the user interface to call createCharacter()
- createCharacter() allows for the user to input their desired information which inputting any piece of data will constantly call inputData()
- inputData() helps store the information desired by the user to then when chosen data is to the users desire, that data is saved to the main method calling saveCharacter().
- The main has the ability to compile the character through compileCharacter() to make sure the character meets requirements to properly update traits.
- When completely and properly compiled a character is created then it is sent back to the user interface for viewing with the ability for the user interface to then call exportCharacter(), which exports the data from the character that was created to a readable file which can be printed out by the interface calling printCharacter() and is sent back to to the user.

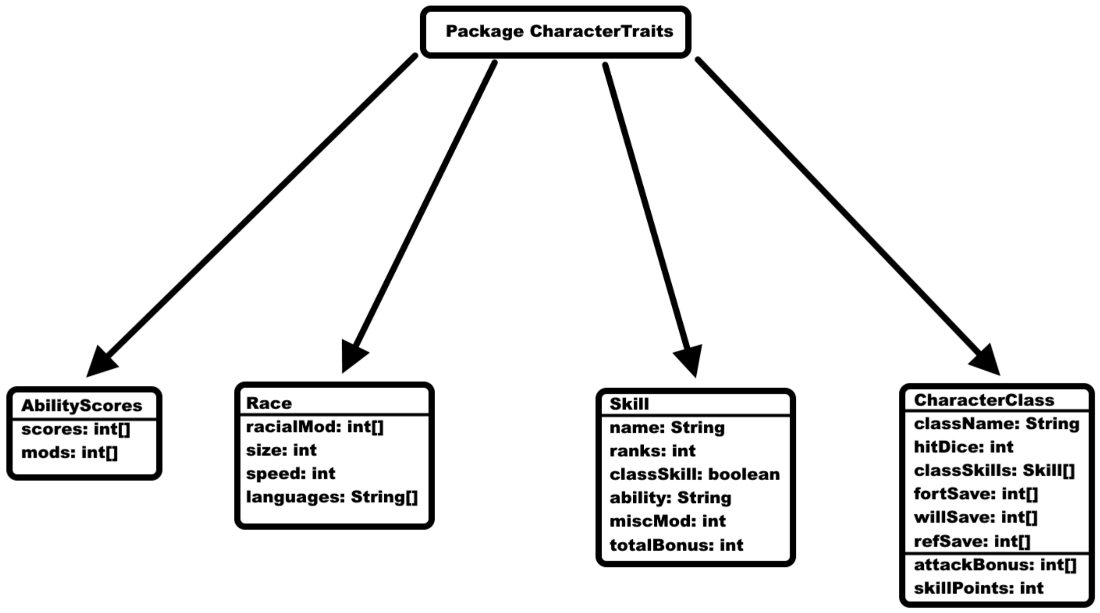
3. Structural Design



Class diagram of system, with all classes, class variables and methods. The CharacterTraits package is not separated into classes, as each class in the package is used by the same classes outside of the package. (A detailed view of the CharacterTraits package can be seen below).



Class diagram for the CharacterTraits package. All of these classes are used primarily as datatypes in the system, and so are all used by every class which references this package.


Package CharacterTraits
AbilityScores
scores: int[]
mods: int[]
Race
racialMod: int[]
size: int
speed: int
languages: String[]
Skill
name: String
ranks: int
classSkill: boolean
ability: String
miscMod: int
totalBonus: int
CharacterClass
className: String
hitDice: int
classSkills: Skill[]
fortSave: int[]
willSave: int[]
refSave: int[]
attackBonus: int[]
skillPoints: int