

# ERIC

## Overview

ERIC is the Eighties Rock Informal Communicator. Eric has a very limited amount of information in his knowledge base. Bands and artists Eric knows are: The Clash, Van Halen, Journey, Tom Petty, Joan Jett & The Blackhearts, AC/DC, Stray Cats, Great White, Dokken, Ozzy Osbourne, Aerosmith, and Def Leppard. Eric knows a few albums these bands put out during the 80s. He knows a few dozen songs from these artists and he has an opinion on every one. Eric also knows the band members and when songs and albums came out. Eric is stuck in the 80s, so he still thinks Steve Perry is the singer in Journey. The only Van Halen album Eric listens to is 1984, so he doesn't acknowledge that Sammy Hagar was the singer after that album. Eric does have a bit of an attitude; if a song sucks, he'll tell you.

1. What is the domain you selected and what approach did you take in solving the problem?

I chose 80s rock as my domain. I decided Eric would know about some bands, albums, songs, band members and the band member's job (singer, guitarist, etc). Since we're trying to pass the Turing test, I decided to give him a personality. If a song sucks, he'll tell you. He can get bored if you keep talking about something other than 80s rock and eventually he'll get annoyed.

I started with a pattern matching approach. I used rules like: (?\* ?X) what's up (?\* ?Y). I just used the strings in parenthesis as literal matches. A pattern can also have no preceding X, like the hello rule. Once I got that working, I made simple rules with a canned response. Then I added multiple responses (Eric randomly picks one). After that I grabbed everything after the match (in other words I grabbed the Y) and stored it in a variable. That way I can ask Why (?\* ?Y). The program is smart enough to end the Y variable at the first period, question mark, or exclamation point.

Once that was working, I worked on the knowledge base. I decided to use a new file for it. The general rules had rules and replies. The knowledge base rules had rule and action. Action was what to look up in the knowledge base. Slowly I built up the types of things it can ask. I then noticed patterns in questions (do you like, do you love, do you hate), so I set up a verbs list, so I could do a "do you (\$Verb)" match. After adding rules to query the database multiple ways, I decided to add "What are you listening to?" and Eric listens to the same thing for about 3 minutes. Finally, I added boredom. Eric gets bored if you talk about too many things unrelated to his knowledge base. I count how many consecutive "non 80s rock" stuff is said by the user. Upon reaching a threshold, Eric asks for you to talk about 80s rock. Once you talk about something in his domain (something that can be answered by the music rules), the "bored counter" resets.

2. Under what situations does your program work well?

You can ask if Eric knows, likes, loves, hates a band, song, or album. You can ask who is the guitarist of a band (or singer or drummer). You can ask things like "Who is Steve Perry?" or ask when a song or album came out. The general rules gives Eric some more conversation flexibility as well as rules with multiple possible answers.

3. Under what situations does your program not work well?

If you ask Eric something he doesn't know, you'll get odd results. For example, you can ask "What songs are on Escape?" (The Journey album from the 80s. Eric knows it and loves it, but he'll give a canned response of "What does it matter?". Eric also doesn't know that band members leave bands. So, if you ask "Who sings in Van Halen?", he'll say David Lee Roth, even though Sammy Hagar was the singer after 1984. I could add all these, but you can always find more stuff to add.

4. Given more time and effort, do you think that you could improve your program using the approach that you tried so that you could eventually build a system that could solve the Turing Test (within the domain you selected)?

I could make it more robust. I could add every 80s rock song, album, band, band member. I could add when someone was a member of the band. I could figure out a lot of combinations of asking questions and querying the data. I could do all that but someone could ask a question in the 80s domain that I didn't consider. For example, Randy Rhodes died in the 80s. It'd be a legitimate question to ask "Is Randy Rhodes alive?" or "How did Randy Rhodes die?". People can ask "Is Van Halen on tour?" Granted, you can simply say "I don't know" to that question but there's a chance you'd get an inappropriate canned response to a question the program didn't understand. People can ask "What kind of guitar did Eddie Van Halen use?" People can ask just about anything related to this domain. So, I don't think I could write a program that would truly solve the Turing test.

5. Anything else you want to say about your program and the experience.

I came up with some interesting ways to connect related data and still maintain Java's quick search for ArrayLists (at least I thought they were interesting). I decided to have an arraylist of matching strings and a similar array list of indexes into other arraylists. For example, when I search for the album "Escape", the program searches through a string arraylist and finds it (let's say it's at index 5. Then the program goes to another array list of album objects and fetches the album at index 5 (of course, it's all of the info about the album "Escape"). I did this quite a bit. Technically, to some degree, my song arraylist supports songs with the same name. I could easily fetch multiple songs of the same name and say what artist sang the songs. Then again, I could have added a lot to it but ran out of time. It does a lot, so I'm relatively happy with the results.

In doing this program, I realized how difficult it is to come up with a program that can talk back and make sense. Consider the word "am". I am Ken. I am tired. I am in the car. One is saying who I am, one is saying my current state, and one is saying where I'm physically located. If someone tells Eric "Do you like Whitesnake?", Eric won't find it and will fall through to the generic rules and may say something that "How about we talk about 80s rock?", even though Whitesnake was an 80s rock band. Eric can't say "I don't know that band" because what if you said "Do you like ice cream?" Granted, ice cream isn't in the domain of "80s rock", however someone could ask an off the wall question like that and a person can handle it. So, the computer should come up with a valid response as well, if it's trying to pass the Turing test.

The next two pages has pieces of conversation with Eric.

## Working conversation

Hey, this is Eric. I love eighties rock.  
I hope you do to, because I prefer to talk about that today.  
So, type something on the keyboard and let's talk.

Hi Eric.

Eric: Hey dude!

Do you like 80s rock?

Eric: What do you think, Einstein?

Who is Steve Perry?

Eric: Steve Perry is the singer of Journey.

Who is the drummer of Journey?

Eric: Steve Smith is the drummer.

Do you like the song Faithfully?

Eric: Faithfully is freakin' awesome!

Do you like Tom Petty?

Eric: Tom Petty is awesome!

Do you like Free Falling?

Eric: Free Fallin' is awesome!

Do you like Def Leppard?

Eric: Def Leppard is ok.

Who is the keyboardist of Def Leppard?

Eric: There isn't a keyboardist in the band.

Do you like the song Rocket?

Eric: Rocket is lame.

Who sings Don't Stop Believing?

Eric: Journey sings Don't Stop Believin'.

Do you like the album Back in Black?

Eric: Back In Black is awesome!

I like ice cream.

Eric: Why do you like ice cream?

It's all cold and creamy.

Eric: How about we talk about 80s rock?

No, I feel like talking about ice cream now.

Eric: If you say so ...

My favorite ice cream is chocolate.

Eric: How about we talk about 80s rock?

Chocolate is the best.

Eric: Okay, enough of the idle chat. Let's talk about 80s rock.

No, I hate 80s rock.

Eric: Don't be dissing my music!

Why not?

Eric: Why do you want to know?

I'm just curious, that's all.

Eric: My God! This is mind numbingly boring!

Bye.

Eric: See you later, dude!

## The program breaks down - #1

Hey, this is Eric. I love eighties rock.  
I hope you do to, because I prefer to talk about that today.  
So, type something on the keyboard and let's talk.

Hey there Eric.

Eric: Hey dude!

Do you like Whitesnake?

Eric: How about we talk about something else?

Do you like Deep Purple?

Eric: If you say so ...

The “If you say so” response should be for statements, not questions. If given time, I could have implemented generic question responses versus generic statement responses and have Eric pick the appropriate one based on punctuation. Of course, if the person used the wrong punctuation, it'll pick from the wrong group.

## The program breaks down - #2

Hey, this is Eric. I love eighties rock.  
I hope you do to, because I prefer to talk about that today.  
So, type something on the keyboard and let's talk.

Hi Eric.

Eric: Hey dude!

Do you like Journey?

Eric: Journey is awesome!

Do you like escape?

Eric: Escape is awesome!

What songs are on Escape?

Eric: What does it matter?

What instrument does Brian Setzer play?

Eric: What does it matter?

I hate rock walls.

Eric: Don't be dissing my music!

How are you?

Eric: I'm cool. Thanks for asking.

How are you able to talk to me?

Eric: I'm cool. Thanks for asking.

There's two break downs here, if you don't count his unwillingness to answer questions.

1. The “I hate rock” rule matches the “I hate rock walls” statement.
2. The “How are you able to talk to me?” matches the “How are you?” rule. I could make it so “How are you” doesn't accept any characters after the match, but then “How are you feeling” and “How are you, Eric?” are legitimate questions that should be answered by the same “How are you?” rule.