Learn Java

# Language Families

Languages, just like people, often have families. You can even map language lineage a bit like a family tree.

Java can help us build a model to track inherited traits across families. In this case, we'll focus on something that often varies between language families: word order — where the subject, verb, and object would go in a sentence.

For example, in English, you would use subject-verb-object order:

| subject | verb | object |
|---|---|---|
| **She** | eats | noodles |

But in Japanese, you would use subject-object-verb order:

| subject | object | verb |
|---|---|---|
| 彼女は | 麺を | 食べる |
| **(she)** | (noodles) | (eats) |

Your **Language Inheritance** package has three Java files:

- **Language.java**: for the `Language` parent class, which serves as the template for all languages.
- **Mayan.java**: for `Mayan`, a child class of `Language` modeled after the Mayan language family.
- **SinoTibetan.java**: for `SinoTibetan`, a child class of `Language` modeled after the Sino-Tibetan language family.

Complete the following 13 tasks to build out a Java package with these classes to model real-world language families.

## A language by any other name (Parent Class)

1. Start with a file **Language.java**, and create a `Language` class with a `main()` method and the following fields:

- `name`: a protected string.
- `numSpeakers`: a protected integer.
- `regionsSpoken`: a protected string.
- `wordOrder`: a protected string.

2. Above the `main()` method, create a `Language` constructor that sets each field to the values passed in.

3. Create a `public` method for `Language` called `getInfo()`. We'll use this method to display all of its information (using its field values).

The method should not return anything.

We want to set up the information like this:

```
[name] is spoken by [numSpeakers] people mainly in
[regionsSpoken].
The language follows the word order: [wordOrder].
```

For example, if we call `spanish.getInfo();`, we'd want to see something like this:

```
Spanish is spoken by 555000000 people mainly in Spain,
Latin America, and Equatorial Guinea.
The language follows the word order: subject-verb-
object.
```

4. Let's test out the code so far!

In `main()`, instantiate a new `Language` (object) of your choice.

Then call `getInfo()` on that new `Language` (object) variable.

Run your code in the terminal to see if the information gets printed. If nothing displays, try compiling your code first.

## Not just an ancient civilization (First Child Class)

5. Nice work! Now let's model a language family.

Now create a new file **Mayan.java** and create an empty `Mayan` class that inherits from `Language`.

6. Mayan languages share several traits in common including:

- `regionsSpoken`: `"Central America"`
- `wordOrder`: `"verb-object-subject"`

Tweak (super constructor to override) the `Mayan` constructor so that it isn't necessary to pass in these fields when instantiating a new `Mayan` language object.

Bear in mind that each language will still require its own `name` and `numSpeakers`.

7. Mayan languages have an interesting grammatical feature: ergativity.

Link: https://en.wikipedia.org/wiki/Ergative%E2%80%93absolutive_language

@Override the `getInfo()` method for `Mayan` so that if we called `getInfo()` an additional line (Fun Fact) will be displayed. For example, with a Mayan language like Ki'che', we'd get the following info:

```
Ki'che' is spoken by 2330000 people mainly in Central
America.
The language follows the word order: verb-object-
subject
Fun fact: Ki'che' is an ergative language.
```

8. Time to test out the tweaks you made to the `Mayan` class…

Go back over to **Language.java**.

In `main()`, instantiate a new `Mayan` language of your choice (you can find one here: https://en.wikipedia.org/wiki/List_of_Mayan_languages).

Then call `getInfo()` on the language variable.

Run your code in the terminal to see if the information gets printed. If nothing displays, try compiling your code first.

## Heading east...(Second Child Class)

9. The Sino-Tibetan family has the second highest number of native speakers of any language family.

Now create a new file **SinoTibetan.java** and build out an empty `SinoTibetan` class that inherits from `Language`.

10. Like the Mayan language family, Sino-Tibetan languages share several traits in common. In this case:

- `regionsSpoken`: `"Asia"`
- `wordOrder`: `"subject-object-verb"`

Build a constructor for `SinoTibetan` so that it isn't necessary to pass in these fields when instantiating a new `SinoTibetan` language object.

Remember — each language will still require its own `name` and `numSpeakers`.

11. Unfortunately, that word order thing? There is actually a split in the Sino-Tibetan family on this (Link: https://en.wikipedia.org/wiki/Sino-Tibetan_languages#Word_order).

It turns out that at some point (a long time ago) the Chinese languages (among a few others) switched the object and verb order. So they now follow a subject-verb-object pattern. Hmm… How can we handle this?

One (imperfect) tactic is to check if the language's `name` field contains `"Chinese"`. There's a Java string method to check if a string contains a substring.

(Link to contains method explanation: https://docs.oracle.com/javase/7/docs/api/java/lang/String.html#contains(java.lang.CharSequence))

In the constructor, below where you called `super()`, change the `wordOrder` to `"subject-verb-object"` if `this.name` contains `"Chinese"`.

## Wrapping up

12. Test out the `SinoTibetan` class by instantiating two new Sino-Tibetan language objects of your choosing:

- One Chinese (e.g., "Mandarin Chinese")
    - https://en.wikipedia.org/wiki/Chinese_language#Grouping
- One non-Chinese (e.g., Burmese)
    - https://en.wikipedia.org/wiki/Sino-Tibetan_languages#Contemporary_languages

Then call `getInfo()` on each language variable.

Run your code in the terminal to see if the information gets printed. If nothing displays, try compiling your code first.

## Where 2 Now?...(Third Child Class)

13. Congrats on all your work with Java Inheritance and Polymorphism! You've built out some useful classes for a linguist out there, but now you are required to extend the project by creating a third child class that extends the parent language class.

There are many more language families you could use and there is a lot more you can do here. It is up to you what family language you choose for your third class. Check the hints below for some ideas to make your Language Inheritance project even better.

Language Families:

https://en.wikipedia.org/wiki/List_of_language_families#Language_families_(non-sign)

Sign Language Families:

https://en.wikipedia.org/wiki/List_of_language_families#Sign_languages