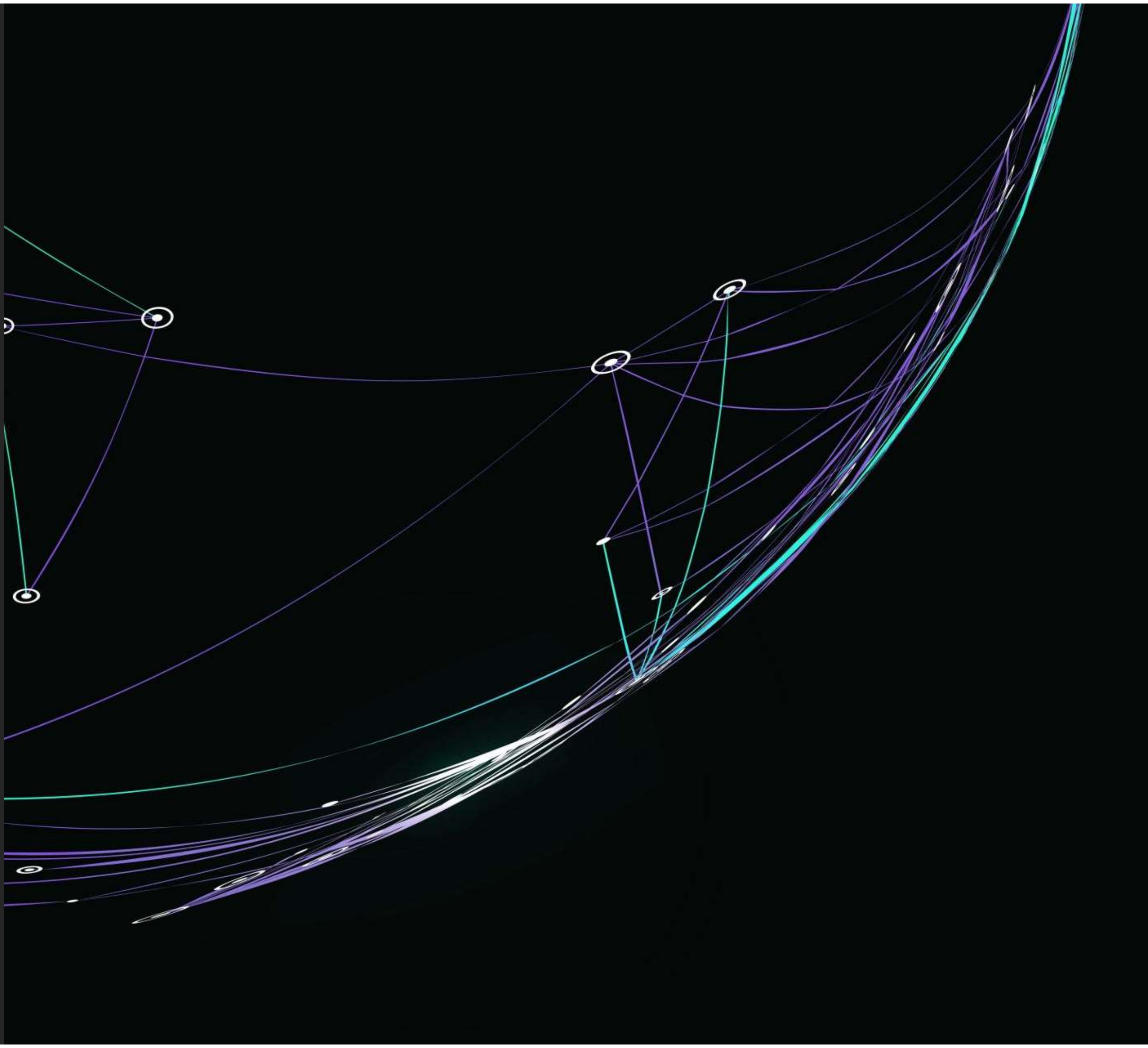
SOFTWARE WITHOUT BORDERS

HOW TO MAKE YOUR PROGRAM A GLOBAL HIT

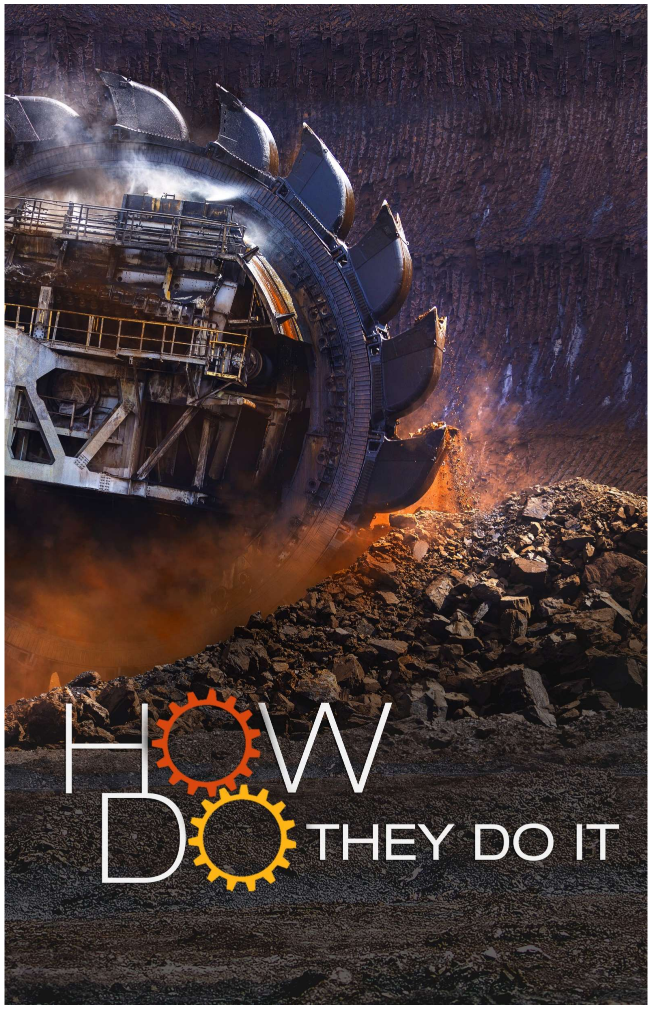


CONTEXT

As a(n) [your country of origin here] developer, I want to be able to develop a game for kids that might/do not speak my native language or a common one like English, Mandarin Chinese or Hindi.

THIS IS A WELL-KNOWN PROBLEM

(and already solved many times)



SO, HOW DO THEY DO IT?

1. Design your program such that it can be easily adapted to different languages and regions (*internationalization* or *i18n*)
2. Use this to actually adapt the software into the target locale (*localization* or *l10n*)

WHAT IS LOCALE SPECIFIC?

- Text
- …?

DESIGN REQUIREMENTS

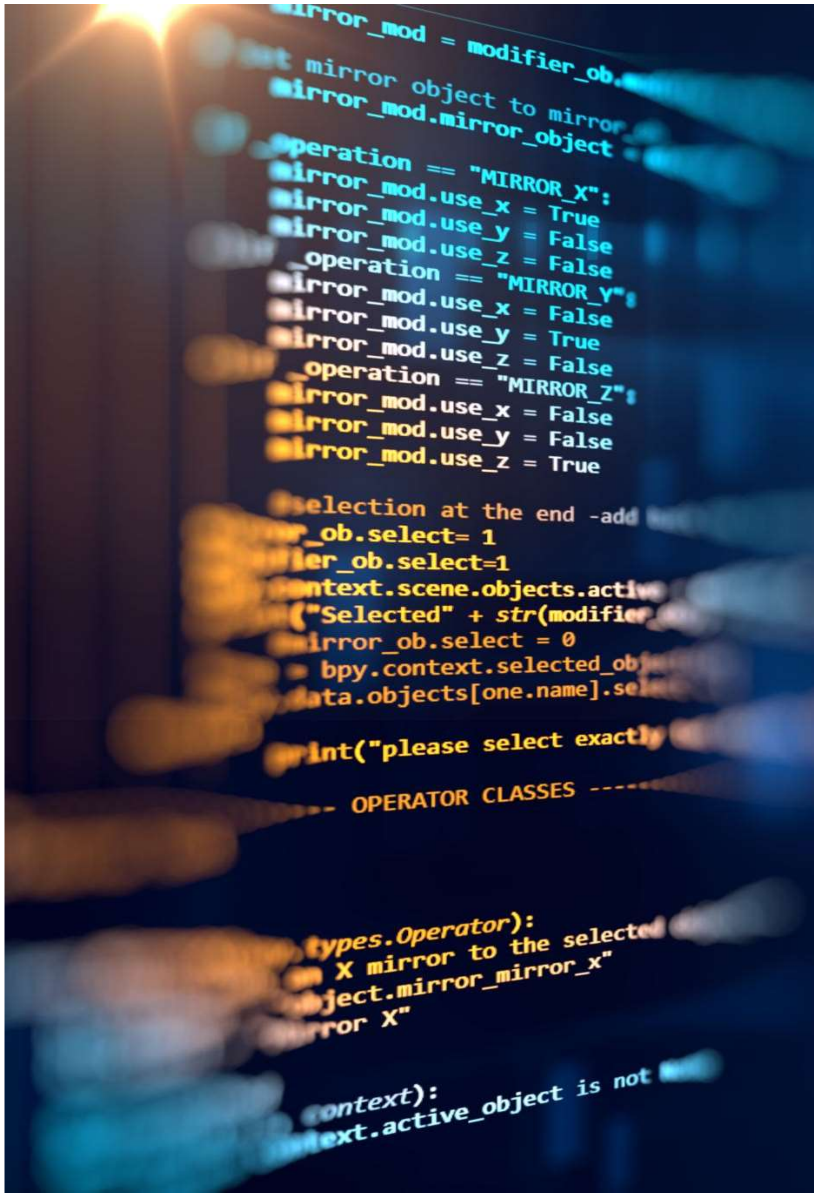
Design a software (part/section/library/module/or whatever you name it) where its users (the programmers that use that software) can…

- Develop an application (game) where the UI is in one given language in a natural way
- Let other people (even non-programmers) translate all language specific resources to different languages
- Let the runtime context (like the application user and/or its OS or browser) set the language at runtime. The user should be able to decide how, and which languages can be selected

SOLUTION PATTERNS

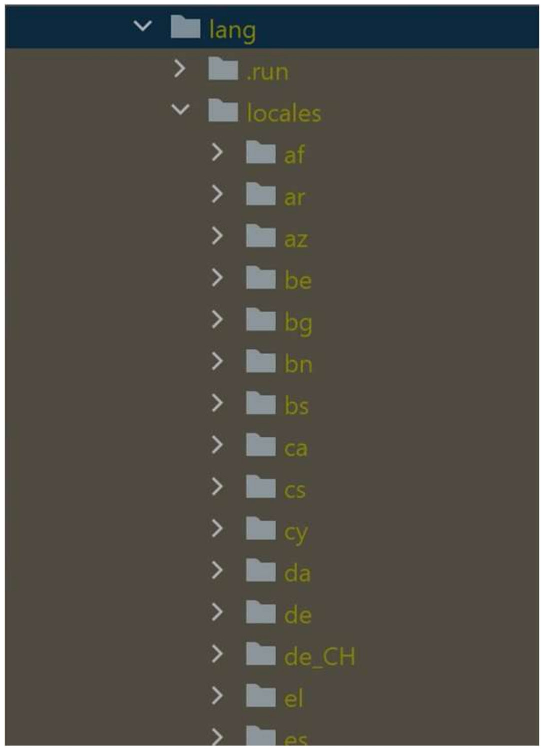
Existing software environments/frameworks/libraries commonly:

- Use some sort of text-based file format to store translation strings
- Make it easy to load these file(s) for a specific language that is set/changed at runtime
- Make it easy to use the translation strings, even with parameters and pluralization
- Make it easy to format dates and numbers



FILE FORMATS

- Text based files. One file (or folder) per language
- Usually structured in a key – value like setting (i.e. `'messages.welcome' => 'Welcome to our application!’`)
- A file format that supports this (like .json, .xml and .yml) is commonly used
- Key also might be the actual string in a default language (i.e. `"I love programming": "Me encanta programar"`)
- File/folder naming usually uses the *RFC 5646* specification



USING TRANSLATION STRINGS

- Usually a meaningful function (like `translate()`) is used. However, super-short names (like `trans()`, `t()` or even `__()`) are also pretty common

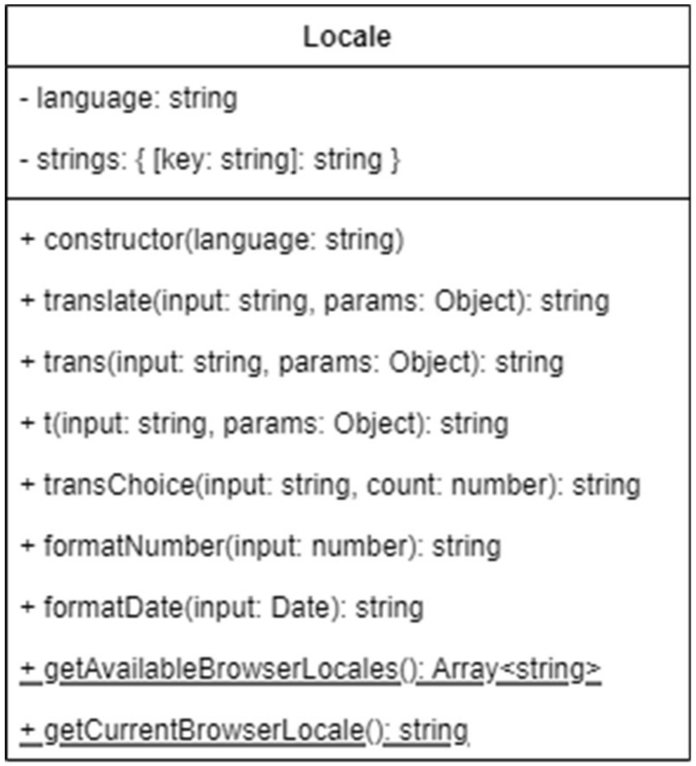
- Strings might be parameterized like: `translate('Hello :name!', { name: 'Eddie' })`

- Pluralization: choose a string pending on some amount

```
const str = '{0} There are no apples|{1} There is one apple|[2,*] There are :count apples’;
transChoice(str, 0));
```

MY (OPINIONATED) TS SOLUTION

- Create a class responsible for all locale specific functionality.
- Each instance of that class should be of a specific locale. This instance should:
  - Load the translation file(s) for that specific language
  - Translate a string, with parameters
  - Translate a string to be chosen, pending on some value
  - Format a number
  - Format a date



Locale
- language: string
- strings: { [key: string]: string }
+ constructor(language: string)
+ translate(input: string, params: Object): string
+ trans(input: string, params: Object): string
+ t(input: string, params: Object): string
+ transChoice(input: string, count: number): string
+ formatNumber(input: number): string
+ formatDate(input: Date): string
+ getAvailableBrowserLocales(): Array<string>
+ getCurrentBrowserLocale(): string

FILE FORMAT

- Files stored in a folder: `assets/lang`
- Chosen for the json file format. File name per RFC 5646 (`nl.json`, `en-US.json`, etc)
- Prefer translation strings as keys:

```
{
  "Hello World": "Hallo Wereld",
  ...
}
```

EXPERIMENT

⚠ Always try to minimize the number of instances created for this class. Loading times can be lengthy

1. Add the Locale class to your project (see gist)
2. Experiment with `Locale.getAvailableBrowserLocales()` `Locale.getCurrentBrowserLocale()`
3. Find a good place (object) to store an instance of that class. Create an instance with language of your choice
4. Find one or more strings to translate
5. ‘Wrap’ the string into a `translate()` method call on that instance
6. Add the lang folder in your assets folder
7. Add a language json file of your choice (or more) to that folder
8. Experiment with translation strings, parameters, pluralization, formatting numbers and dates