2022/07/21 課題 村上航暉

課題 1-1

> 以下の手順を実行して、動作を確認してください。
>
> 1. src ディレクトリを対象にするよう、tsconfig.json を修正してください。

```
{
  "compilerOptions": {
    ……
  },
  "include": ['./src']
}
```

> 2. ルートディレクトリの直下に src ディレクトリを作成してください。
> 3. src ディレクトリに index.ts を作成してください。
> 4. index.ts に以下の内容を記載してください。

```
console.log('hello world');
```

> 5. この時点で、src ディレクトリに JS ファイルが存在するか確認してください。
> 6. ターミナルを開き、「 npx tsc -w 」または「 tsc -w 」を実行してください。 → 「Watching for file changes.」という文言が表示されることを確認
> 7. VS Code の Explorer タブをリフレッシュするか、ターミナルを使って、src ディレクトリに index.js ファイルが生成されているか確認してください。
> 8. index.ts ファイルの内容を変更してください。
> 9. ファイルの更新後、src/index.js ファイルの内容を確認し、TS ファイルと同じように内容が更新されているか確認してください。
> 10. src ファイルの外の TS ファイル（例: hello.ts）を修正し、内容が更新されるか確認してください。

手順	確認結果
手順 5	存在しない
手順 7	生成されている
手順 9	更新されていない (tsc コマンド実行後は更新される)
手順 10	更新されない (tsc コマンド実行後も更新されない)

課題 1-2

> 以下は CommonJS 形式の モジュールのインポートです。 ES モジュールの構文で index.ts を記述してください。

```
export function greetings(name: string) {
  console.log(`hello, ${name}`);
}
```

```
import util = require('./util');
util.greetings('hoge');
```

`.\src\index.ts` にて実装。

```
import { greetings } from './util';
greetings('hoge');
```

課題 2-1

> 『Node.js & TypeScript のプロジェクト作成』の内容を踏まえて、以下の手順を実行した際にどうなるかを確認してください。
>
> 1. tsconfig.json を別名で退避する
> 2. ターミナルから『Node.js & TypeScript のプロジェクト作成』のとおりに tsc --init を再実行する
> 3. src ディレクトリに TS ファイルを用意する（例: index.ts）
> 4. ターミナルから 3 で用意した TS ファイルをコンパイルする（例: tsc src/index.ts）
> 5. ターミナルから npm run buil を実行する（npx tsc でも可）

手順	確認結果
手順 2	新たに `tsconfig.json` が生成された
手順 4	`.\src\project.js` が生成された
手順 5	`.\lib\` に js ファイルが生成される `.\src\` に無い `*.ts` がある場合エラー文が出る

```
node ➜ /workspaces/20220721_traning (main ✗) $ npm run build

> 20220721_traning@1.0.0 build
> tsc -p .

error TS6059: File '/workspaces/20220721_traning/hello.ts' is not under 'rootDir'
'/workspaces/20220721_traning/src'. 'rootDir' is expected to contain all source
files.
  The file is in the program because:
    Matched by include pattern '**/*' in 'tsconfig.json'
```

```
Found 1 error.

node ➜ /workspaces/20220721_traning (main X) $
```

課題 2-2

> TypeScript: TSConfig リファレンス - すべての TSConfig のオプションのドキュメント または tsconfig.json の全オプションを理解する（随時追加中） - Qiita を参照し、生成された tsconfig.json で有効になっている内容をまとめてください。

オプション	値	内容
lib	es6, dom	ES2015と DOM 型のライブラリをコンパイルに使用する
target	es2016	ES2016準拠で js ファイルを出力する
module	commonjs	import/export の記述は CommonJS 準拠で js ファイルを出力する
rootDir	src	出力ディレクトリの構造 はsrcの構造に合わせる
resolveJsonModule	true	*.jsonをモジュールとしてインポートできる
outDir	lib	libを出力ディレクトリとして指定
esModuleInterop	true	CommonJSとES Modulesの両方でモジュールが使用できる記述で js ファイルを出力する
forceConsistentCasingInFileNames	true	import のファイルパスで大文字 or 小文字を区別する
strict	true	strict*の型チェック設定をすべて有効にする
skipLibCheck	true	型定義ファイルをチェックしない 型定義のコピーが複数存在してしまってる時等に有効

課題 3-1

> 以下のミックスイン用の関数を使って、Animal を拡張したクラスを作ってください。 また、クラスからインスタンスを生成して、各メソッドが実行できることを確認してください。

```ts
type Constructor<T = {}> = new (...args: any[]) => T;

/** 走る */
function Runnable<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    run() {
      console.log('Running!');
    }
  };
}
```

```
/** 鳴く */
function Cryable<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    cry() {
      console.log('Cring!');
    }
  };
}

/** 飛ぶ */
function Flyable<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    fly() {
      console.log('Flying!');
    }
  };
}

class Animal {}
```

.\src\animal.ts にミックスイン用関数を実装。 .\src\hawk.ts, .\src\tiger.ts, .\src\locust.ts にて実装。