Lab 5: Type manipulation

TypeScript's type system is very powerful because it allows expressing types in terms of other types.

Union types

A union type describes a value that can be one of several types, in this section we'll create a new type that can be either a string or a number:

1. Open the starter project with VSCode
2. Open `models.ts` file and create a new type `ID` which combine `number` and `string` types

   ```
   export type ID = number | string
   ```

3. Update the type annotation for `id` property in the `Task` inteface

   ```
   id: ID;
   ```

4. Open data.ts file and update the type annoation for the `id` parameter for the CRUD functions `addTask`, `updateTask` and `deleteTask`

   ```
   function addTask(id: ID, ...)
   function updateTask(id: ID, ...)
   function deleteTask(id: ID, ...)
   ```

5. Run `tsc`

Intersection types

An intersection type combines multiple types into one. This allows you to add together existing types to get a single type that has all the features you need:

1. Open `models.ts` file
2. Declare three types for each priority of the task: `UnimportantTask`, `ImportantTask` using intersection operator

```
export  type ImportantTask = Task & {priority: Priority}
export  type NormalTask = Task & {priority: Priority}
export  type UnimportantTask = Task & {priority: Priority}
```

3. In Typescript we can use values as types, in this case we can specify the value priority property even without variable assignement like the following

```
export  type ImportantTask = Task & {priority: Priority.High}
export  type NormalTask = Task & {priority: Priority.Normal}
export  type UnimportantTask = Task & {priority: Priority.Low}
```

> By doing so we avoid the risk of having conflicts between types, like assigning a low priority value for an `ImportantTask` type

4. Add another type `TaskWithPriority` which groups all of the types above (using union)

```
export  type TaskWithPriority = ImportantTask | NormalTask | UnimportantTask
```

5. Refactor `data.ts` so that it uses `TaskWithPriority` type instead of `Task` type

6. Run `tsc` to type-check

Type transformation

Indexed access type

We can use an indexed access type to look up a specific property on another type:

Open `data.ts` file and update the return type of `addTask` and `updateTask` so it uses the type of the array `tasks`, to do so you can use indexed access type on the array type

```
    export  function addTask(...): typeof tasks[number]
    export  function updateTask(...): typeof tasks[number]
```

> `typeof tasks[number]` is simply `Task`, we extracted the type that used with the generic type `Array<T>`

Utility types

TypeScript provides several utility types to facilitate common type transformations. These utilities are available globally.

1. Open `models.ts` and remove the `assignedTo` property of the `Task` low priority tasks, i.e. for the types `UnimportantTask` Use the utility type `Omit<T, Keys>` to remove specific `Keys` from `T` type:

```
    export  type UnimportantTask = Omit<Task, "assignedTo"> & {priority: Priority.Low}
```

2. Create a variable `notImportant` of type `UnimportantTask` and assign an object instance to check for type errors.

3. Run `tsc`