САМАРСКИЙ УНИВЕРСИТЕТ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Лекция № 2

Структуры данных. Массивы. Кортежи. Дженерики. Перечисления. Кастомные типы

2023

РУБРИКА: «ВОПРОСЫ ПО ЛЕКЦИИ № 1»

Предисловие о структуре данных

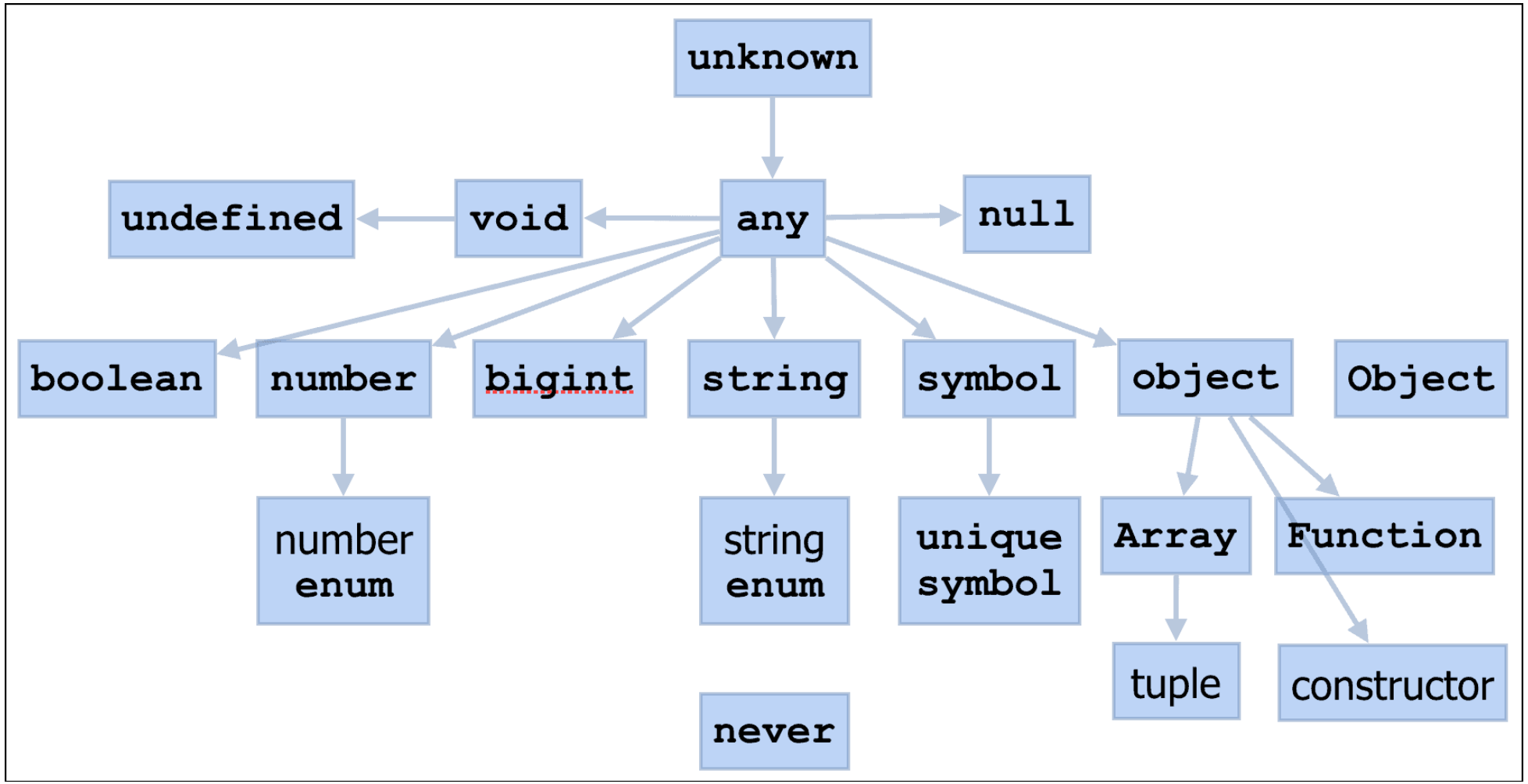
Что такое Структура данных?

Это сведения о строении некоторой модели данных, над которым может выполнять действия реализованное программное обеспечение.

Зачем это нужно?

Именно с этой целью и был изобретен TypeScript, а именно для реализации упрощения построения структур данных и их ведения в процессе дополнения ПО новыми функциональными возможностями, их модификации, оптимизации и так далее... (повышение читабельности кода, повышение скорости разработки, модификации, повышение эффективности поиска ошибок в коде).
В чистом JavaScript структуры данных не имеют валидации и пишутся вслепую, так как отсутствует компиляция, от чего могут следовать неявные ошибки, которые могут попасть в ПО к конечному пользователю.

Кастомные типы как частный случай компоновки примитивов



Что такое кастомный тип?

Это тип, строение и функции которого характеризуются другими кастомными типами (вложенными) и, если не имеется сведений о применяемых кастомных типах – примитивами (string, number, boolean, и т.д...).

Примеры кастомных типов в TypeScript

```
interface Client {
  id: number;
  name: string;
  surname: string;
}
```

```
interface Employee {
  id: number;
  name: string;
  surname: string;
  role: RolePart1 & RolePart2;
}
```

```
enum Season {
Winter = "Зима",
Spring = "Весна",
Summer = "Лето",
Autumn = "Осень "
};
```

```
type AccountInfo: Client | Employee;
```

Недостатки JavaScript?

— Отсутствие модульности — могут возникать проблемы из-за конфликта больших файлов.
— Нелогичное поведение.
— Динамическая типизация — нет IntelliSense, из-за чего мы не видим, какие ошибки будут возникать во время написания, а видим их только во время исполнения программы

Достоинства TypeScript?

— Статическая типизация.
— Компилируется в «хороший» JavaScript.
— Легко читаемый код, который понятен с первого взгляда.
— Делает язык приближенным к языкам общего назначения

Что такое массив?

Это структура данных, реализующая возможность ведения некоторого определенного множества объектов одинакового типа в рамках адресации к каждому элементу этого массива по индексу

Как это адресация по индексу?

Представьте множество домов, стоящих на одной улице. Номер дома – это его индекс. Для того, чтобы обратиться к сведениям о доме, мы должны точно знать его номер (индекс)

Как быть, если улиц много и много городов?

Для этого достаточно объявить, что ссылки на номер дома недостаточно для однозначного определения сведений о нём (домов с одинаковым номером в городе очень много) и необходимо также знать ссылку на улицу, по которому определенный дом с определенным номером находится. Для городов делаем аналогично, добавляя в начале еще один индекс перед улицей и номером.

Создадим некоторый интерфейс, характеризующий дом

```
interface Building {
  appartmentsNumber: number;
  entrancesNumber: number;
  managerInfo: string;
}
```

Создадим объект, содержащий множество домов

```
const streetBuildings0: Building[] = [] //Пустой массив

const streetBuildings2: Building[] = [ //Заполненный массив
  {
    appartmentsNumber: 120,
    entrancesNumber: 6,
    managerInfo: “Васюткина Лидия Михайловна”,
  },
  {
    appartmentsNumber: 60,
    entrancesNumber: 3;
    managerInfo: “Иванов Петр Васильевич”,
  },
];
```

Для обращения к элементу массива нужно указать его порядковый номер следования:

```
const streetBuilding: Building = streetBuildings2[0]
//=> {
      appartmentsNumber: 120,
      entrancesNumber: 6,
      managerInfo: “Васюткина Лидия Михайловна”,
    }
```

Структуры данных :: Массивы :: Пример двумерного массива

```
const streetGararina: Building[] = [
   {
      appartmentsNumber: 120,
      entrancesNumber: 6,
      managerInfo: “Васюткина Лидия Михайловна”,
   },
   {
      appartmentsNumber: 60,
      entrancesNumber: 3;
      managerInfo: “Иванов Петр Васильевич”,
   },
];
```

```
const streetNovosadovaya: Building[] = [
   {
      appartmentsNumber: 180,
      entrancesNumber: 10,
      managerInfo: “Петрова Мария Николаевна”,
   },
   {
      appartmentsNumber: 50,
      entrancesNumber: 2;
      managerInfo: “Рожков Роман Иванович”,
   },
];
```

```
const streets: Building[][] = [
 streetGararina, streetNovosadovaya
];

console.log(streets[0][1]);
```

```
[LOG]: {
  "appartmentsNumber": 60,
  "entrancesNumber": 3,
  "managerInfo": "Иванов Петр
Васильевич"
}
```

Что такое кортеж?

Это упорядоченный набор фиксированной длины.

Как это кортеж?

Представьте управляющего в вашем доме. Вы знаете его ФИО, № квартиры, в котором он проживает и его номер телефона. Дополнительная информация о нем вам и не нужна, поэтому достаточно создать некоторый набор данных, в котором достаточно легко понять, где ФИО, где адрес, а где номер телефона

Что такое дженерик?

Это неявно определенный тип до момента компиляции программного кода, позволяющий сократить количество написанного кода для подобных структур данных и их обработчиков, таким образом привнося в исходный код программы абстракцию и некоторую «демократию»

Зачем они нужны?

Представьте транспортное средство...
Вот оно... Кто-то подумал об автобусе, кто-то о легковом автомобиле, а кто-то о велосипеде, самокате и так далее.
Представим ситуацию, что абсолютно любое транспортное средство в некотором государстве подлежит учету. У каждого экземпляра транспортного средства есть как общие признаки (наличие уникального номера рамы или кузова), так и различия (например автомобиль содержит номер двигателя, а велосипед его не имеет вовсе).
Для того, чтобы определить свойства как можно эффективнее, мы наследуем от некоторого абстрактного интерфейса Транспортное средство (Далее ТС) все его свойства и расширим его под конкретный тип ТС.
Допустим, нам понадобилась функция, возвращающая общие сведения о ТС, причем для каждого типа ТС формат отображения будет одинаковым. Именно в этом моменте могут помочь Дженерики.

Создадим некоторый метод, который может просто возвращать тоже самое, что ему и передали в параметре

```
function identity<T> (param: T): T {
  return param;
}
```

Можно также использовать дженерики для построения пользовательских структур

```
interface EmployeeDepartment<D extends Department>
{
  id: number;
  fio: string;
  phone: string;
  department: D
}
```

```
interface Department {
  id: number;
  name: string;
  manager: string;
}

interface ITDepartment extends Department {
  teamLeader: string;
}

interface SalesDepartment extends Department {
   priceController: string;
}
```

Таких параметров может быть много и они могут быть зависимы друг от друга

function identity<T, D> (param1: T, param2: D): T //независимые дженерики

function identity<T, D extends keyof T> (param1: T, param2: D): D //А здесь передаваемый тип D зависим от ключа типа T. т.е. что зададим тип для 1го, то повлияет на решение о допустимости 2го типа, что впоследствии будет обработано компилятором и в случае наличия несоответствия приведет к ошибке при компиляции.

Что такое перечисления?

Это некоторое конечное множество чего-либо в определенной предметной области, не подвергающийся модификации в процессе исполнения некоторой функции этой самой предметной области.

Зачем они нужны?

Допустим у нас есть Яблони в саду. Мы знаем сорт каждого посаженного дерева и новых сортов сажать не собираемся.
Нам необходимо сделать ПО учета яблоневого сада, в котором мы будем учитывать каждое дерево. Но вот незадача, программисты не знали, какие сорта у нас высажены и нам приходится вести учет, внося данные о сорте каждому дереву вручную без возможности выбора и проверки, что в этом поле написано.
Со временем часто забывающий очки дома садовник допустил множество ошибок при внесении данных о деревьях, и было решено зафиксировать сорта в строгий список – перечисление. Теперь у нас есть только выбор сорта без возможности внесения собственного.

```
enum SystemRole {
   ADMIN, USER, MANAGER, CALL_CENTER, DEVOPS, DEVELOPER, TESTER
}
```

```
enum Season {     Winter = "Зима",     Spring = "Весна",    Summer = "Лето",
Autumn = "Осень"};
```

```
enum Season {     Winter = 1,     Spring = "Весна",    Summer = 3,     Autumn =
"Осень"};
```

```
enum SystemRole {
  ADMIN, USER, MANAGER, CALL_CENTER, DEVOPS,
DEVELOPER, TESTER
}
```

Как строго ограниченное множество для выбора конкретного значения(-й) свойства объекта

```
interface User {
   id: number;
   ...
   role: SystemRole;
}
```

Как аргумент метода

```
const countByRole = (role: SystemRole): number => {...}

function countByRole(role: SystemRole): number {...}
```

РУБРИКА: «ВОПРОСЫ СЛУШАТЕЛЕЙ»

КОНЕЦ ЛЕКЦИИ № 2
СПАСИБО ЗА ВНИМАНИЕ