Summary

When to use generics

Type parameters

Generic functions, classes, and interfaces

Adding constraints to generic classes

Demo

Adding a constraint to a generic class

```
nterface CatalogItem {
   catalogNumber: number;

lass Catalog<T extends CatalogItem> implements Inventory<T> {
   // implement interface methods here
```

Generic Constraints

Describe types that may be passed as a generic parameter

“extends” keyword applies constraint

Only types satisfying the constraint may be used

```
nterface CatalogItem {
   catalogNumber: number;

lass Catalog<T extends CatalogItem> implements Inventory<T> {
   // implement interface methods here
```

Generic Constraints

Describe types that may be passed as a generic parameter

“extends” keyword applies constraint

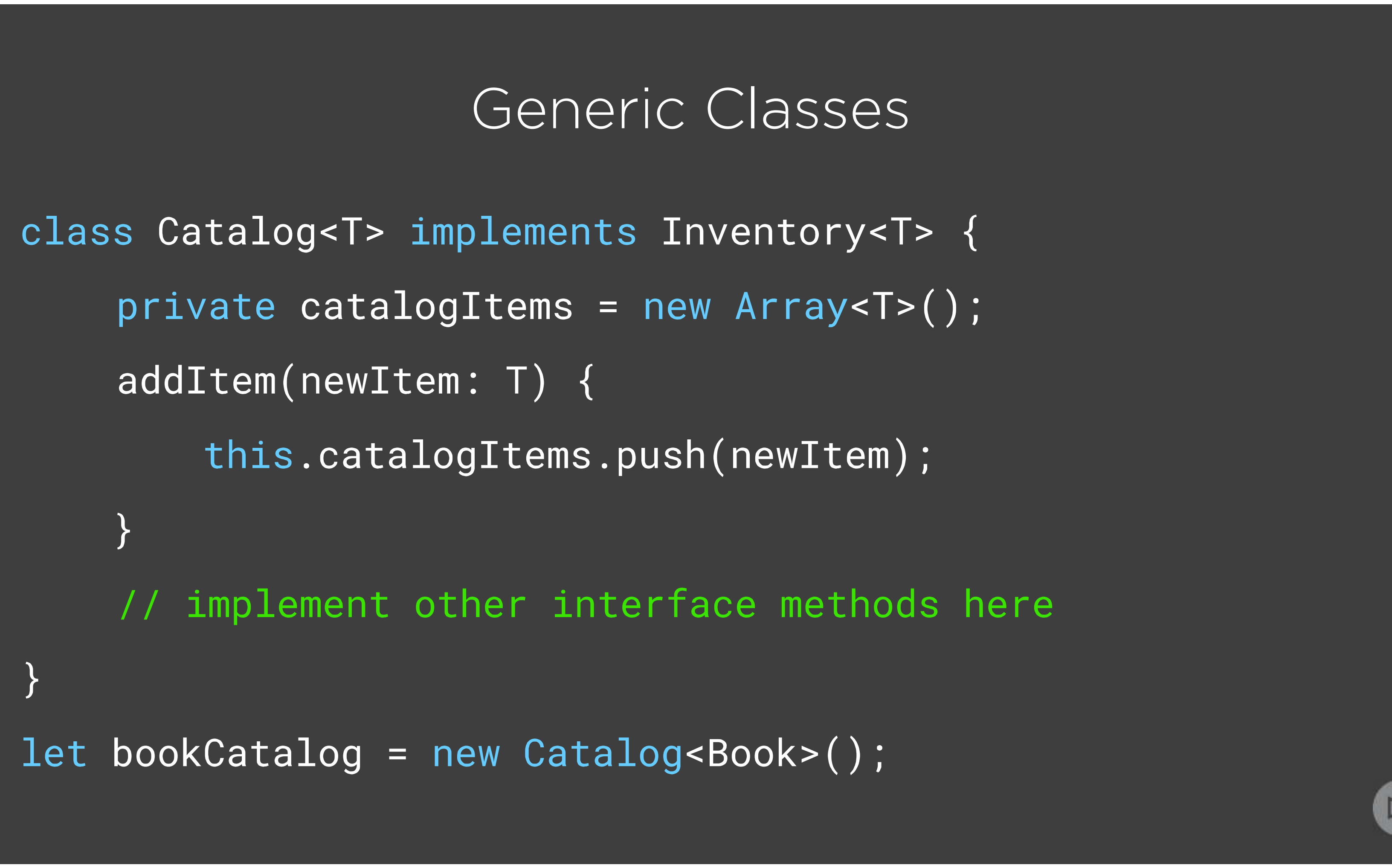
“I’m a real believer in that creativity comes from limits, not freedom.”

Jon Stewart

Fresh Air (NPR)
Jon Stewart: The Most Trusted Name In Fake News

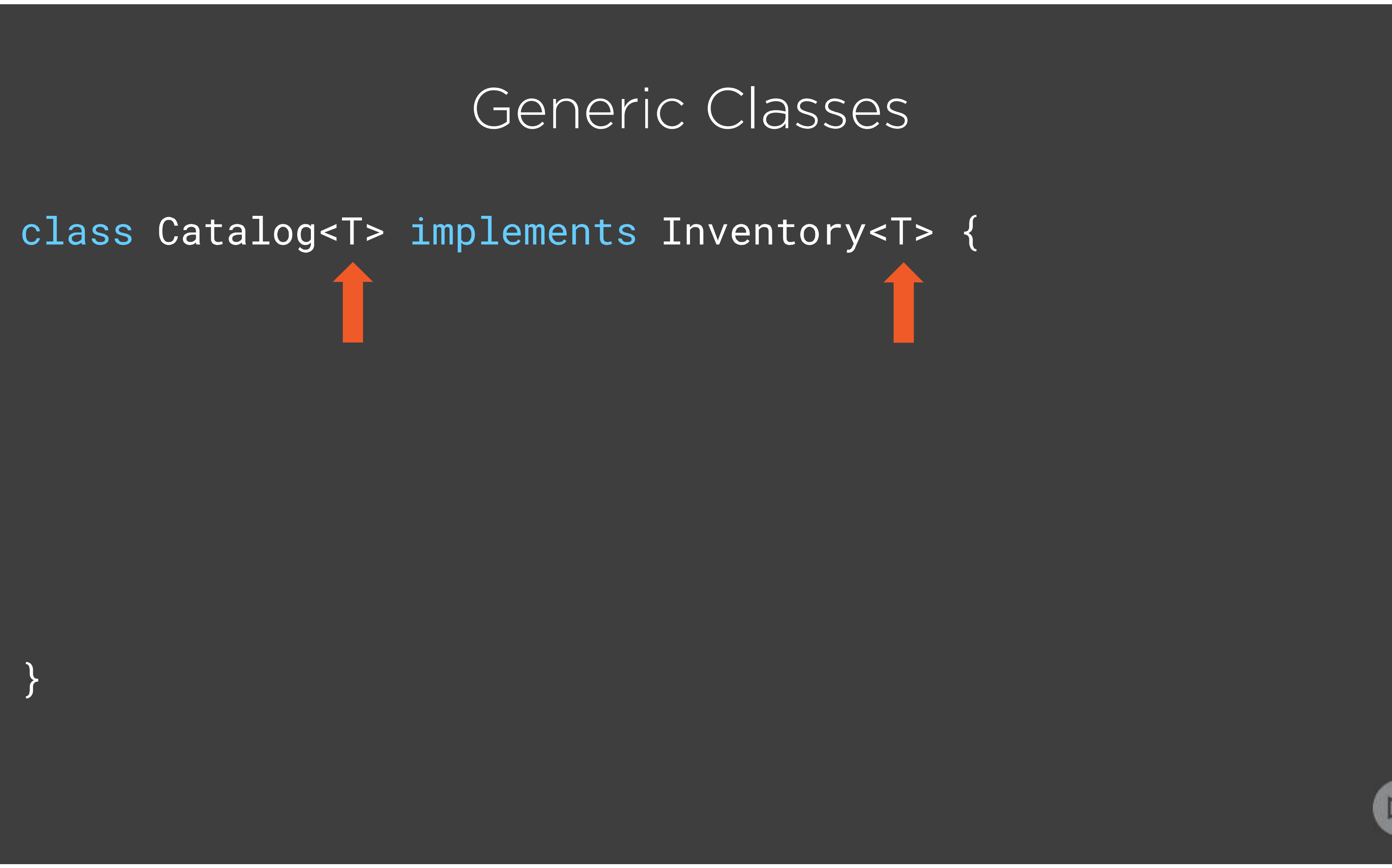
Demo

Creating and using a generic class



Generic Classes

```
class Catalog<T> implements Inventory<T> {
     private catalogItems = new Array<T>();
     addItem(newItem: T) {
         this.catalogItems.push(newItem);
     }
     // implement other interface methods here
}
let bookCatalog = new Catalog<Book>();
```

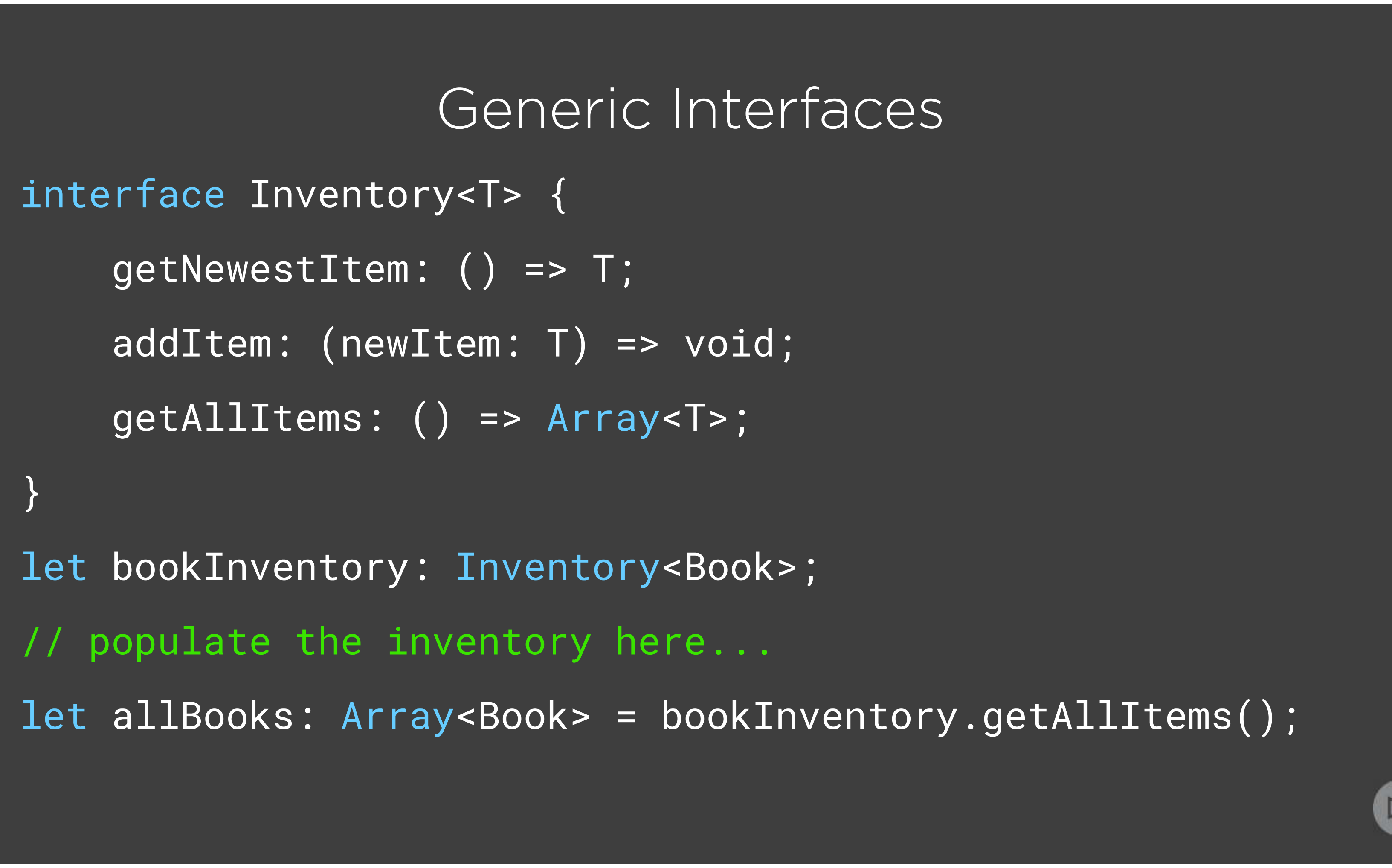


Generic Classes

```
class Catalog<T> implements Inventory<T> {



}
```



Generic Interfaces

```
interface Inventory<T> {
    getNewestItem: () => T;
    addItem: (newItem: T) => void;
    getAllItems: () => Array<T>;
}
let bookInventory: Inventory<Book>;
// populate the inventory here...
let allBooks: Array<Book> = bookInventory.getAllItems();
```

Generic Interfaces

```
interface Inventory<T> {




}
```

Demo

Creating and using generic functions

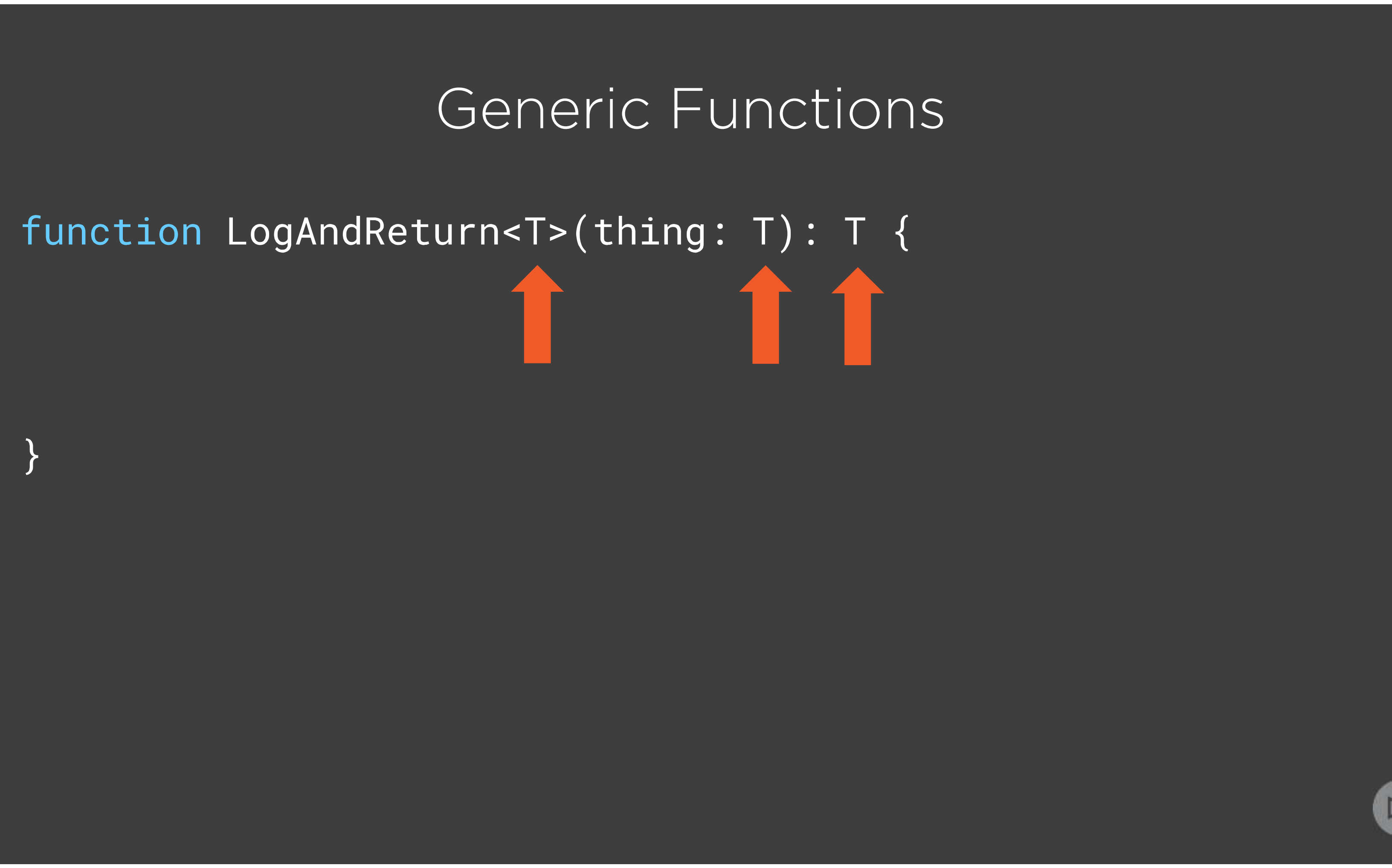
Generic Functions

```
function LogAndReturn<T>(thing: T): T {
    console.log(thing);
    return thing;
}
let someString: string = LogAndReturn<string>('log this');

let newMag: Magazine = { title: 'Web Dev Monthly' };
let someMag: Magazine = LogAndReturn<Magazine>(newMag);
```

Generic Functions

```
function LogAndReturn<T>(thing: T): T {
    console.log(thing);
    return thing;
}
let someString: string = LogAndReturn<string>('log this');
```



Generic Functions

```
function LogAndReturn<T>(thing: T): T {



}
```

```
let poetryBooks: Book[];

let fictionBooks: Array<Book>;

let historyBooks = new Array<Book>(5);
```

Using Array<T>

Type parameter specifies the type the array can contain

Type parameters are part of the type

Type parameters are listed separate from function parameters

```
let poetryBooks: Book[];

let fictionBooks: Array<Book>;
```

Using Array<T>

Type parameter specifies the type the array can contain

Type parameters are part of the type

What Are Type Parameters?

Specify the type a generic will operate ove

Listed separate from function parameters inside angle brackets

Conventionally represented by the letter ‘T (e.g. Array<T>)

Actual type provided at instance creation or function invocation

What Are Generics?

Code that works with multiple types

Accept “type parameters” for each instance or invocation

Apply to functions, interfaces, and classes

Overview

What are generics?

Type parameters

Generic functions

Generic classes and interfaces

Generic constraints