```
Pair<string, Entity>
```

is indistinguishable from the type

```
{ first: string; second: Entity; }
```

3.5.1 Instance Types

Each named type has an associated actual type known as the ***instance type***. For a non-generic type, the instance type is simply a type reference to the non-generic type. For a generic type, the instance type is an instantiation of the generic type where each of the type arguments is the corresponding type parameter. Since the instance type uses the type parameters it can be used only where the type parameters are in scope—that is, inside the declaration of the generic type. Within the constructor and instance member functions of a class, the type of `this` is the instance type of the class.

The following example illustrates the concept of an instance type:

```
class G<T> {                // Introduce type parameter T
    self: G<T>;             // Use T as type argument to form instance type
    f() {
        this.self = this;   // self and this are both of type G<T>
    }
}
```

3.6 Specifying Types

Types are specified either by referencing their keyword or name, by querying expression types, or by writing type literals which compose other types into new types.

Type:
    PredefinedType
    TypeReference
    TypeQuery
    TypeLiteral

3.6.1 Predefined Types

The `any`, `number`, `boolean`, `string`, and `void` keywords reference the Any type and the Number, Boolean, String, and Void primitive types respectively.

PredefinedType:
    `any`
    `number`
    `boolean`
    `string`
    `void`

The predefined type keywords are reserved and cannot be used as names of user defined types.