Type systems & Generics

CO5225 – Andrew Muncey

Contents

Type Safety

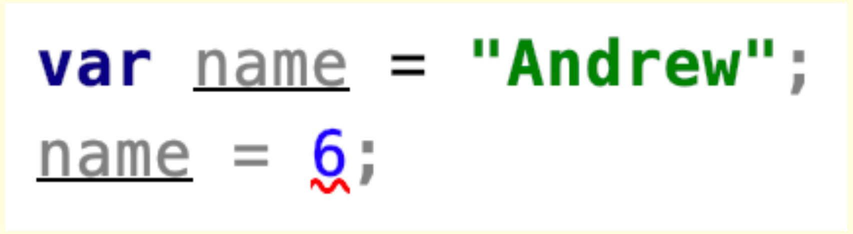
- How well a programming language prevents type errors
- A type error can occur when one data type is unintentionally treated as another

Statically / dynamically typed language

- A statically typed language verifies the type safety of of a program at compile time
- A dynamically typed language resolves type at run time

Changing type

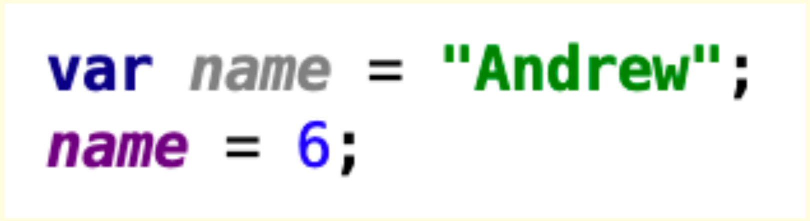
Kotlin (Static)



```
var name = "Andrew";
name = 6;
```

Invalid

JavaScript (Dynamic)

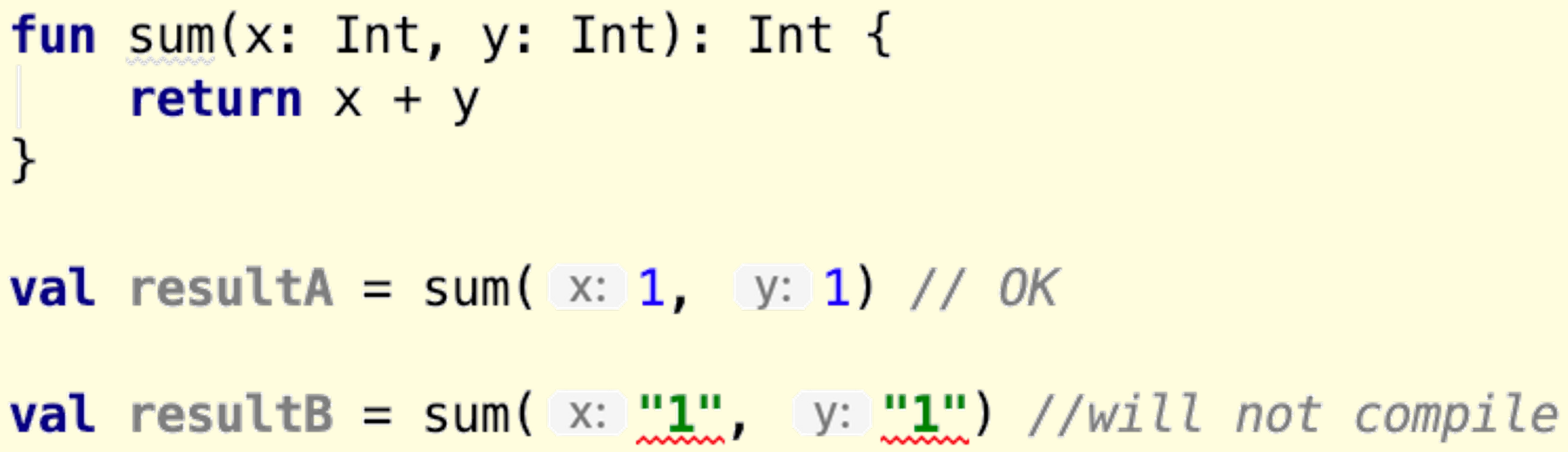


```
var name = "Andrew";
name = 6;
```

Valid

Statically typed language, e.g. Kotlin

- Cannot compile if types do not match



```
fun sum(x: Int, y: Int): Int {
    return x + y
}

val resultA = sum( x: 1,  y: 1) // OK

val resultB = sum( x: "1",  y: "1") //will not compile
```

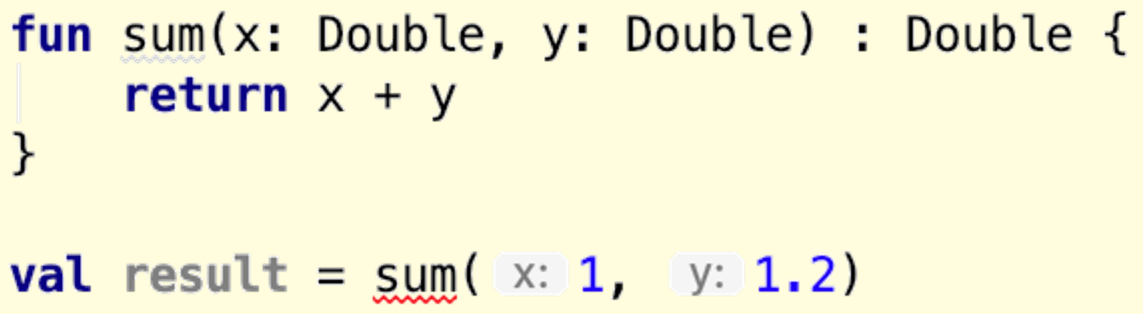
Dynamically typed language, e.g. JavaScript

- Doesn’t care about types at compilation

```
function sum(x, y){
  return x + y;
}

var resultA = sum(1,1); //OK =2
var resultB = sum("1", "1"); // ="11"
var resultC = sum(1, "1"); // ="11"
var resultD = sum(1, 2.3); // =2.33
```

Statically typed languages: Kotlin vs Java

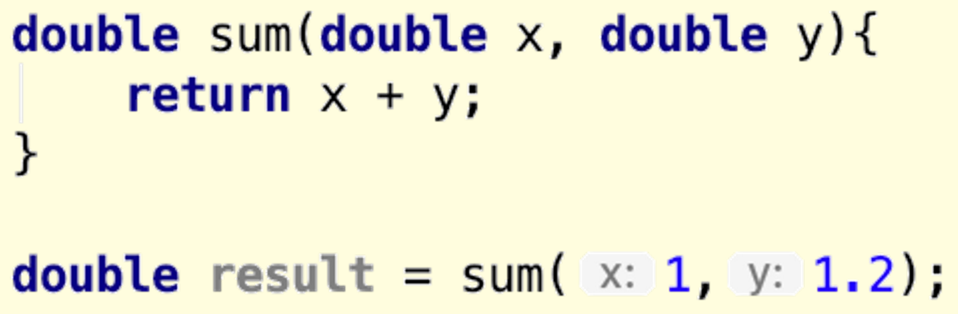
Kotlin



```kotlin
fun sum(x: Double, y: Double) : Double {
    return x + y
}

val result = sum( x: 1,  y: 1.2)
```

Does not compile

Java



```java
double sum(double x, double y){
    return x + y;
}

double result = sum( x: 1, y: 1.2);
```

Compiles

Questions to consider

1. Do statically typed languages have to declare the type of every variable?
2. How flexible is each language with changing the type of a variable after it has been declared?

Which is better (for the developer)?

- Probably static, but not clear cut (Hanenberg, 2010, Okon & Hanenberg 2016)
  - Static types may decrease application complexity and therefore decrease development time
  - Dynamic types may allow the developer to express themselves in a less constrained way, and therefore decrease development time
  - “...strengthens the results of previous findings that actually found measurable benefits for statically typed languages”

Strong / Weak typed languages

- Limited definition of ‘strongly-typed languages’
- “Whenever an object is passed from a calling function to a called function, it’s type must be compatible with the type declared in the called function” (Liskov & Zilles. 1974. p.57)
- Strongly typed languages more likely to be Statically typed

- Might consider how flexible/safe languages are with certain types, e.g.,
  - Can you add a double to an int without explicit conversion?
  - What about implicit conversion between say byte and int?

Generics

- Generics help us work around the constraints of a statically typed language, whilst maintaining type safety
- Consider the concept of a list, we might want to constrain what type of items we put into the list
- The below indicates the type of errors that can occur if we don’t constrain types in collections

Python: Runtime error

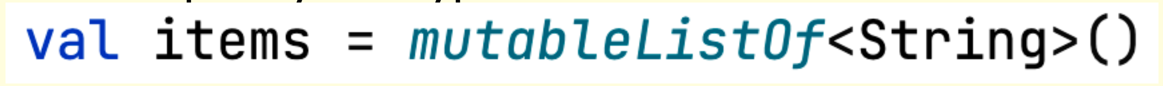
```python
my_list = ["Apple", "Pear", 7]

for item in my_list:
    print(item.upper()) #errors on '7'
```

Kotlin: Compile error

```kotlin
val items = mutableListOf("Apple", "Banana", 7)
for (item in items){
    println(item.uppercase())
}
```

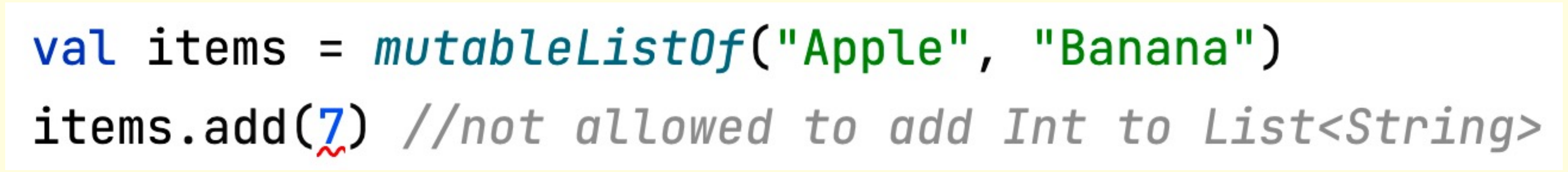
Generics

- In Kotlin we normally wouldn't mix types when creating a list, as we did in the previous slide
- We can specify the type



```kotlin
val items = mutableListOf<String>()
```

- Or the type can be inferred, preventing future type errors

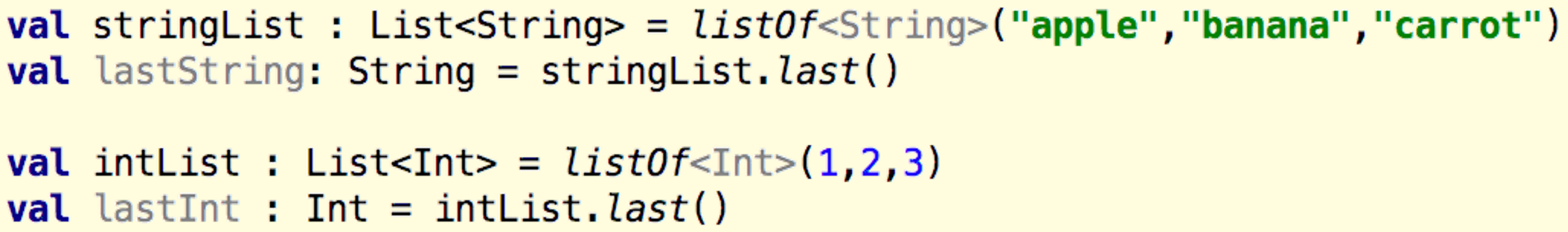


```kotlin
val items = mutableListOf("Apple", "Banana")
items.add(7) //not allowed to add Int to List<String>
```

- List / MutableList are Generic types. Each has a parameter, indicated by Angular brackets in which the type is specified
- A generic type instantiated with a type parameter, i.e., List<String> is known as a parameterized type
- In the previous slide's example, the MutableList's type would be inferred as 'Any'
  - Kotlin recognise that both types implement comparable and Serializable, so methods belonging to those types could be used

Generics

- Consider this code



```
val stringList : List<String> = listOf<String>("apple","banana","carrot")
val lastString: String = stringList.last()

val intList : List<Int> = listOf<Int>(1,2,3)
val lastInt : Int = intList.last()
```

- What is the return type of List’s last() method?