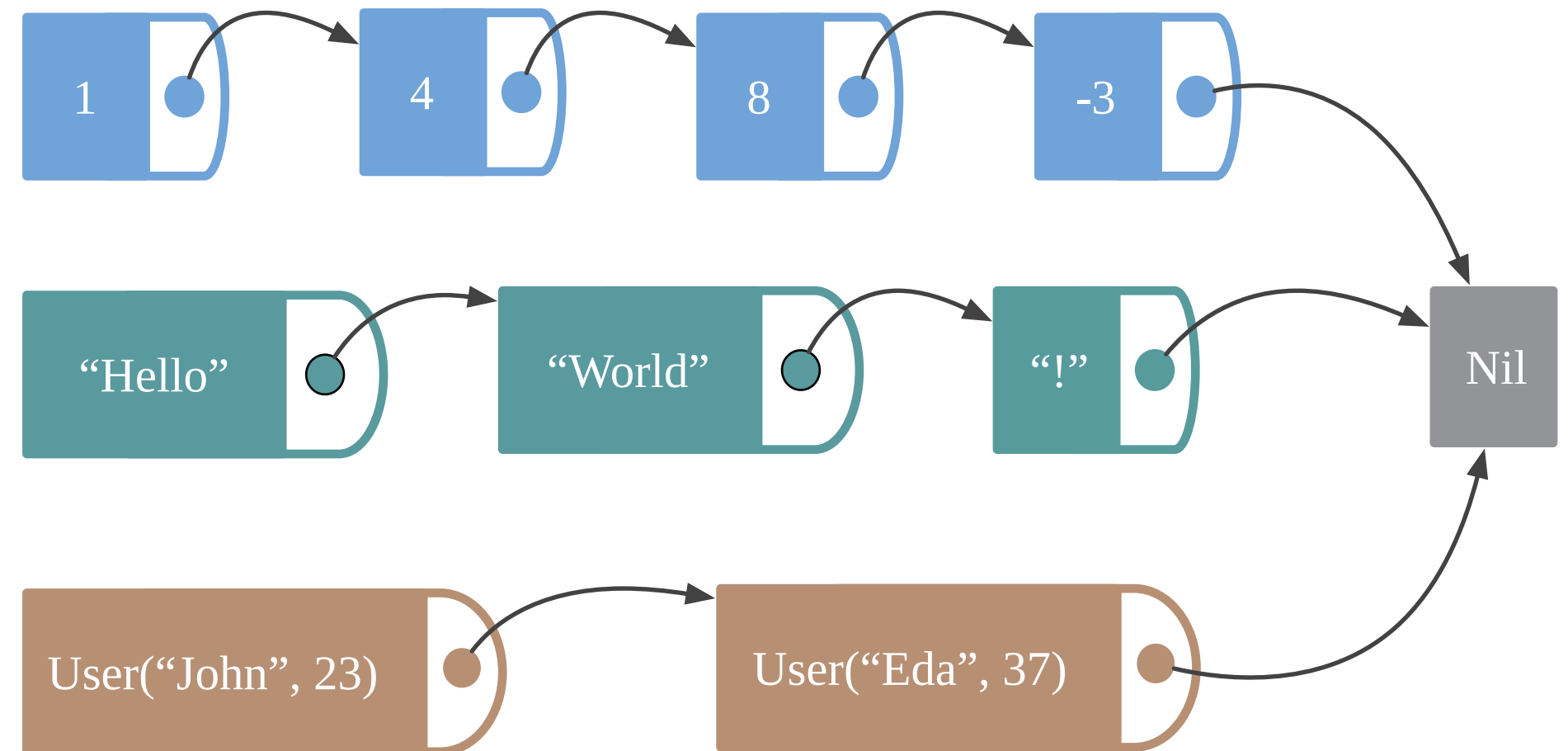




GENERIC FUNCTIONS PART 1

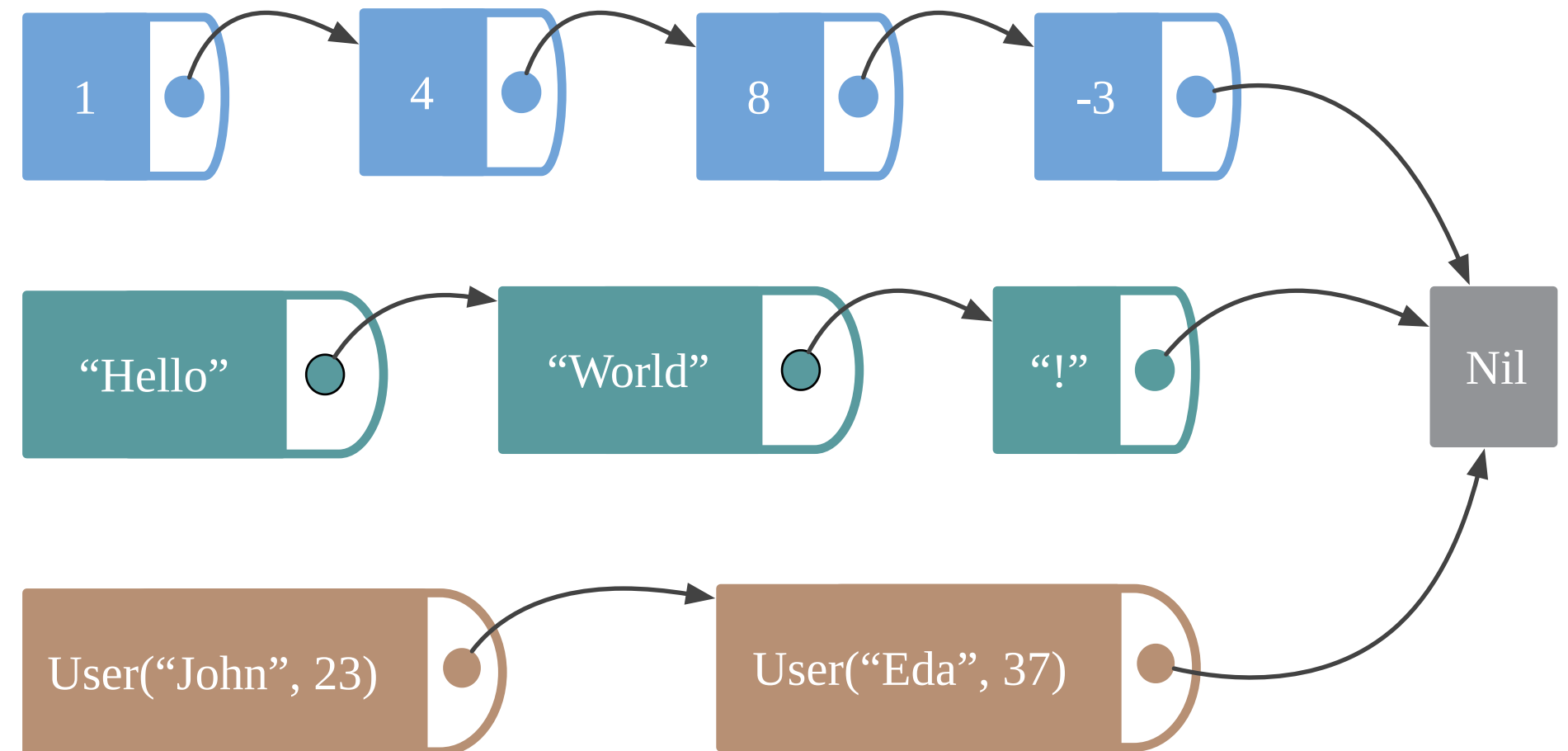
List is a generic data structure

```
List[Int]  
List[String]  
List[User]
```



How to avoid duplication?

```
def size(list: List[Int]): Int  
def size(list: List[String]): Int  
def size(list: List[User]): Int
```

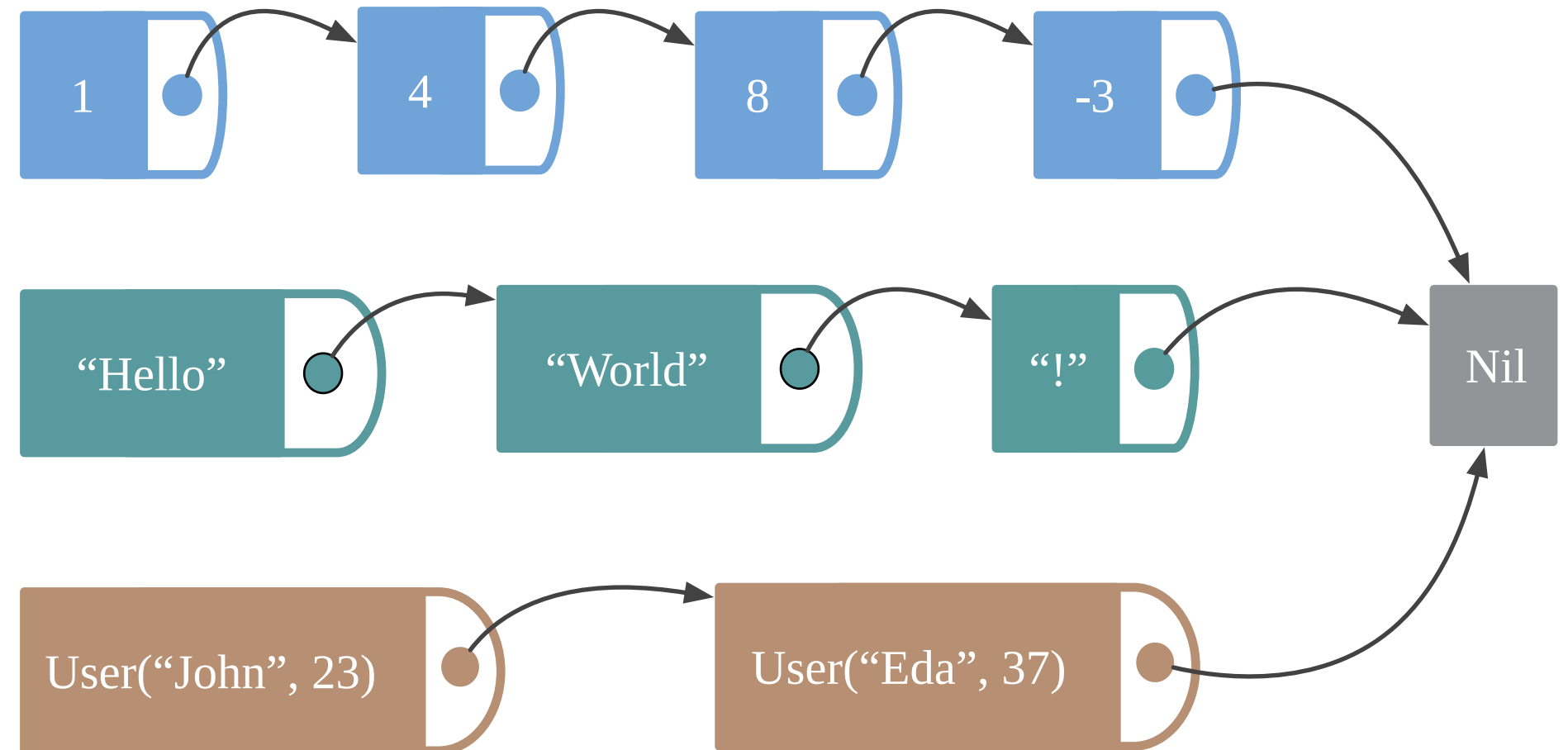


Generic function

```
def size[A](list: List[A]): Int
```

```
size(List(1, 4, 8, -3))  
// res1: Int = 4
```

```
size(List("Hello", "World", "!"))  
// res2: Int = 3
```

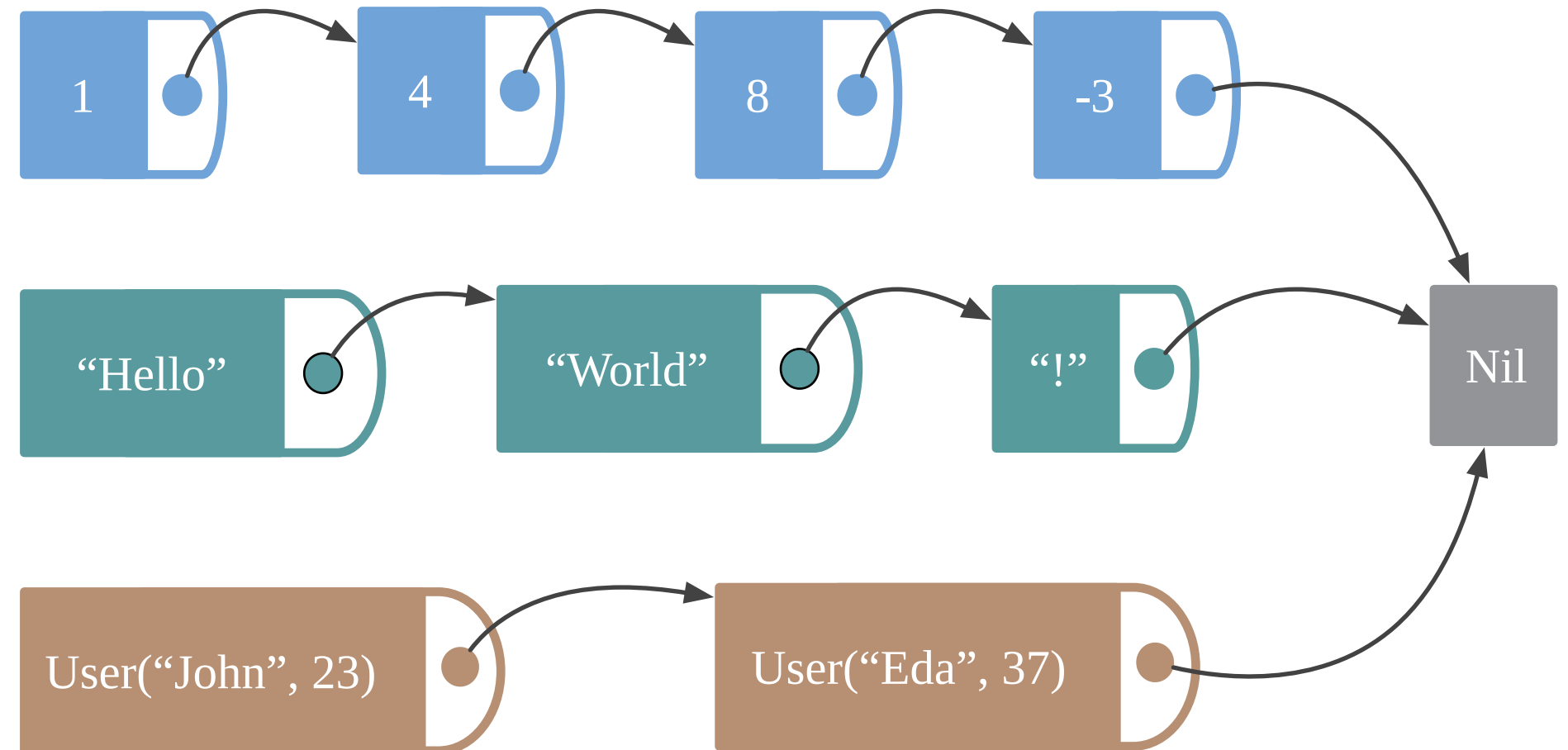


Generic function

```
def size[Elem](list: List[Elem]): Int
```

```
size(List(1, 4, 8, -3))  
// res3: Int = 4
```

```
size(List("Hello", "World", "!"))  
// res4: Int = 3
```



Generic function

```
def map[A](list: List[A], update: A => A): List[A]
```

```
map(List(1,2,3,4), (x: Int) => x + 1)  
// res5: List[Int] = List(2, 3, 4, 5)
```

```
map(List("Hello", "World"), (x: String) => x.reverse)  
// res6: List[String] = List("olleH", "dlrow")
```

Generic function

```
def map[A](list: List[A], update: A => A): List[A]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users, (user: User) => user.name)
// error: type mismatch;
// found   : App0.this.User => String
// required: java.io.Serializable => java.io.Serializable
// map(users, (user: User) => user.name)
//           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Generic function

```
def map[A](list: List[A], update: A => A): List[A]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map[User](users, (user: User) => user.name)  
// error: type mismatch;  
// found    : String  
// required: App0.this.User  
// map[User](users, (user: User) => user.name)  
//                               ^^^^^^^^^^^
```


Generic function

```
def map[From, To](list: List[From], update: From => To): List[To]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users, (user: User) => user.name)  
// res10: List[String] = List("John", "Eda", "Bob")
```

```
map(List(1,2,3,4), (x: Int) => x + 1)  
// res11: List[Int] = List(2, 3, 4, 5)
```

Generic function

```
def map[From, To](list: List[From], update: From => To): List[To]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users, user => user.name)
// error: missing parameter type
// map(users, user => user.name)
//           ^^^^

map(users, _.name)
// error: missing parameter type for expanded
// function ((<x$1: error>) => x$1.name)
// map(users, _.name)
//           ^
```

Generic function

```
def map[From, To](list: List[From])(update: From => To): List[To]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users)(user => user.name)  
// res14: List[String] = List("John", "Eda", "Bob")
```

```
map(users)(_.name)  
// res15: List[String] = List("John", "Eda", "Bob")
```

Generic function in Dotty/Scala 3

```
def map[From, To](list: List[From], update: From => To): List[To]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users, user => user.name)  
// res16: List[String] = List("John", "Eda", "Bob")
```

```
map(users, _.name)  
// res17: List[String] = List("John", "Eda", "Bob")
```

All generic types are not data structure

```
trait JsonDecoder[A]{  
  def decode(value: Json): A  
}  
  
case class Predicate[A](value: A => Boolean)
```

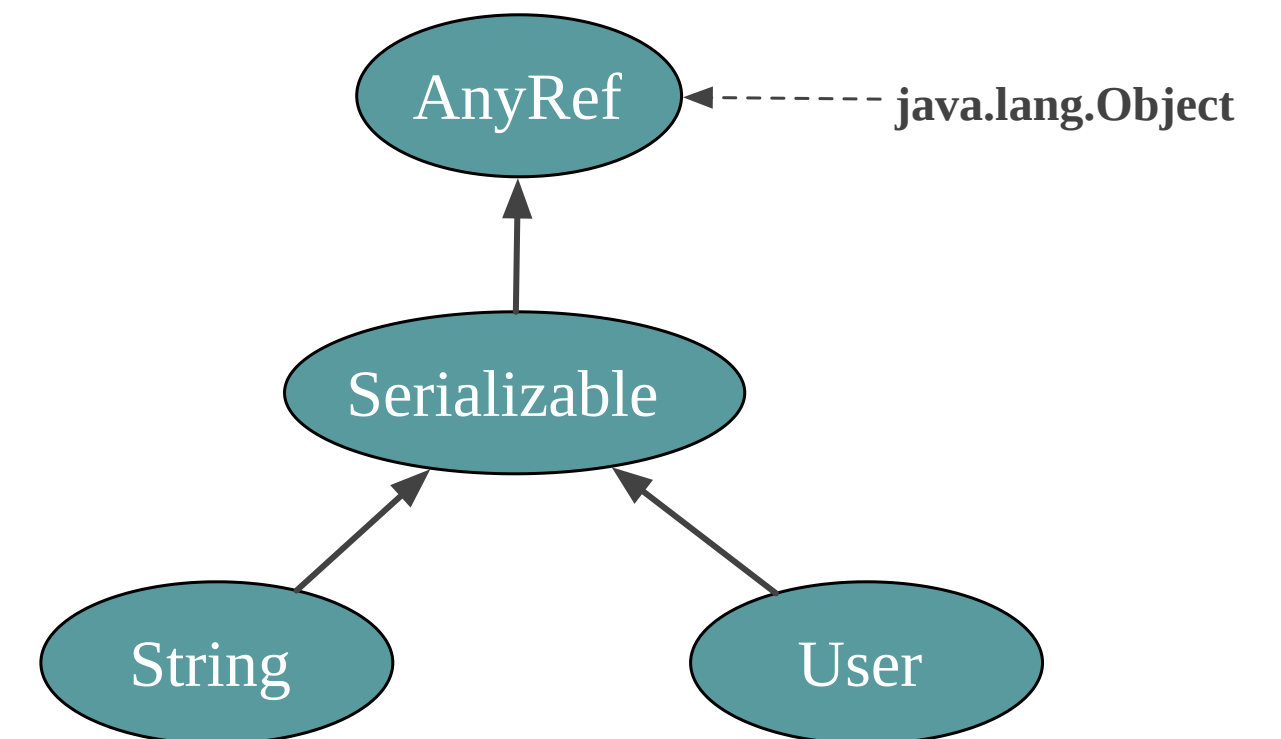
Summary

- Generic functions reduce code duplication
- We can have several type parameters
- We split parameters into different set of parentheses to help type inference

Least Upper Bound (LUB)

```
def map[A](list: List[A], update: A => A): List[A]
```

```
map(users, (user: User) => user.name)
// error: type mismatch;
// found   : App5.this.User => String
// required: java.io.Serializable => java.io.Serializable
// map(users, (user: User) => user.name)
//      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```



Generic functions in Dotty

```
def map[A](list: List[A], update: A => A): List[A]
```

```
val users = List(User("John", 23), User("Eda", 37), User("Bob", 18))
```

```
map(users, (user: User) => user.name)
// Type Mismatch Error:
// map(users, (user: User) => user.name)
//           ^^^^^^^^^^^
//           Found:    (user.name : String)
//           Required: User
```