

Introduction to Generics in Java

Motivation Example- 1

- Need to deal with a dynamic collection of Dog's in more than one context
- How to solve this?
- Reuse code with Composition

```
public class DogCollection {  
    private Dog[] _dogs;  
    private int _size;  
  
    public DogCollection(int initialSize) { ... }  
  
    private void resize() { ... }  
  
    public void add(Dog d) { ... }  
  
    public boolean remove(Dog d) { ... }  
  
    public Dog get(int idx) { ... }  
}
```

Motivation Example - 2

- Same needed functionality with Cat's
- Solution
- Problem?
 - Duplication of code
 - Both classes are almost equal!

```
public class CatCollection {  
    private Cat[] _cats;  
    private int _size;  
  
    public CatCollection(int initialSize) { ... }  
  
    private void resize() { ... }  
  
    public void add(Cat c) { ... }  
  
    public boolean remove(Cat c) { ... }  
  
    public Cat get(int idx) { ... }  
}
```

Motivation Example- 3

- Define a generic collection of Objects
- Advantage
 - Can hold anything
- Disadvantage?
 - Casting
 - Type checking

```
public class Collection {  
    private Object[] _collection;  
    private int _size;  
  
    public Collection(int initialSize) { ... }  
  
    private void resize() { ... }  
  
    public void add(Object o) { ... }  
  
    public boolean remove(Object o) { ... }  
  
    public Object get(int idx) { ... }  
}
```

Disadvantage: Casting

- With DogCollection

```
DogCollection dogs;  
dogs = new DogCollection(5);  
Dog d = new Dog("Cão");  
dogs.add(d);  
Dog myDog = dogs.get(0);
```

- With generic Collection

1. Collection dogs;
2. dogs = new Collection(5);
3. Dog d = new Dog("Cão");
4. dogs.add(d);
5. Dog myDog = dogs.get(0);

Which line is wrong?

- A. None
- B. Line 1
- C. Line 4
- D. Line 5

Need a cast at line 5

(Dog)dogs.get(0);

Disadvantage: Type Checking

- With DogCollection

1. DogCollection dogs;
2. dogs = new DogCollection(5);
3. Cat c = new Cat("Tareco");
4. dogs.add(c);
5. Dog myDog = dogs.get(0);

What happens?

- Compilation error at line 4

- With generic Collection

1. Collection dogs;
2. dogs = new Collection(5);
3. Cat c = new Cat("Tareco");
4. dogs.add(c);
5. Dog myDog = (Dog)dogs.get(0);

What happens now?

- No compilation error
- But execution error
 - At line 5
 - ClassCastException

Better Solution: Generic Types

- Java has syntax for *parameterized data types*
 - Referred to as *Generic Types* in most of the literature
- A *generic type* is a generic class or interface that is parameterized over types
 - Can have one or more *type parameters*
 - *Each type parameter represents a data type*
- When a generic type is instantiated have to specify a type for each parameter type

Generic Types

- Data type parameters declared in class/interface header after name inside < >

```
public interface Collection<E> {
```

- The <E> is the declaration of a data type parameter for the class
 - any legal identifier: `Foo`, `AnyType`, `Element`, `DataTypeThisListStores`
 - Java style guide recommends terse identifiers
 - The data type parameter can be used inside the generic type to refer to a type
- The value E stores will be filled in whenever a programmer declares a variable of type `Collection`

```
Collection<String> li = new Collection<String>(30);
```


Example – Generic Collection

```
public class Collection<E> {  
    private E[] _collection; // not exactly true  
    private int _size;  
  
    public Collection(int initialSize) { ... }  
  
    private void resize() { ... }  
  
    public void add(E o) { ... }  
  
    public boolean remove(E o) { ... }  
  
    public E get(int idx) { ... }  
}
```

1. `Collection<Dog> dogs;`
2. `dogs = new Collection<Dog>(5);`
3. `dogs = new Collection<>(5);`
4. `Dog d = new Dog("Cão");`
5. `dogs.add(d);`
6. `dogs.add(new Cat());`
7. `Dog myDog = dogs.get(0);`

Generics, Inheritance, and Subtypes

- Given Animal as superclass of Dog and Cat
- Is Collection<Animal> a super type of Collection<Cat> and Collection<Dog>?

• **NO!**

```
public class Collection<E> {  
    ...  
    public void add(E o) { ... }  
}
```

Collection<Animal>.add(Animal)

```
Collection <Dog> dogs = new Collection<>(30);  
Collection<Animal> animals = dogs; ❌
```

animals.add(new Dog()); ✓

animals.add(new Cat()); ✓

- Problem with code?
 - Dogs now holds two objects, a dog and a **cat**
 - Compilation error for ***animals = dogs***

Restrictions on Generics

- Cannot instantiate generic types with primitive types
- Cannot create instances of type parameters
- Cannot declare static fields whose types are type parameters
- Cannot use casts or *instanceof* with parameterized types
- Cannot create arrays of parameterized types
- Cannot create, catch, or throw objects of parameterized types