05. Java Generics

Java Generics

- Generics means parameterized types.
- Enables to create classes, interfaces, and methods in which the type of data upon which they operate is specified as a parameter.

Java Generics

- Java Generics is a powerful addition to the Java language because it makes the programmer's job easier and less error-prone.

- Generics enforce type correctness at compile time and, most importantly, enable implementing generic algorithms without causing any extra overhead to our application

Why Generics?

- The functionality of Gen class can be achieved without generics by specifying Object type and using proper casting whenever required

Then why we use Generic?

- Java compiler does not have knowledge about the type of data actually stored in NonGen So:
  - Explicit casts must be employed to retrieve the stored data
  - Several type mismatch errors cannot be found until run time

Why Generics?

- Stroger type checks at compile time
- Elimination of casts

```
ArrayList list = new ArrayList();
list.add("hello");
String s = (String)list.get(0);
```

- Using generics:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);//no cast
```

General form of Generic

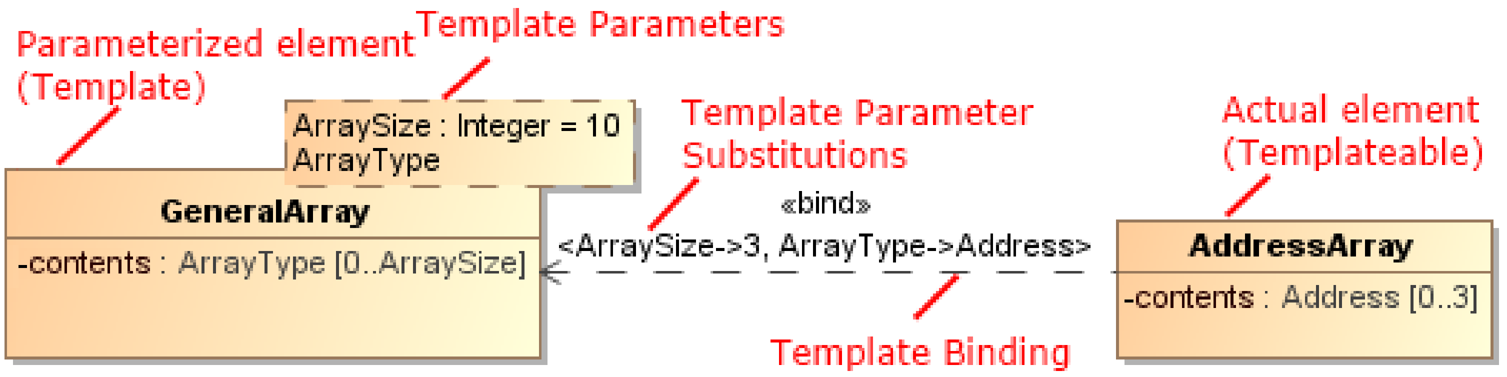
- The generics syntax for declaring a generic class:

```
class class-name<type-param-list>
                    { //… }
```

- The syntax for declaring a reference to a generic class:

```
class-name<type-arg-list> var-name = new class-
                              name<type-arg-list>(cons-arg-list);
```

UML


Parameterized element
(Template)
Template Parameters
ArraySize : Integer = 10
ArrayType
GeneralArray
-contents : ArrayType [0..ArraySize]
Template Parameter
Substitutions
«bind»
<ArraySize->3, ArrayType->Address>
Actual element
(Templateable)
AddressArray
-contents : Address [0..3]
Template Binding

Problem

- create a generic class that contains a method that returns the average of an array of numbers of any type, including integers, floats, and doubles.

Wildcards generics

- Wildcards help in allowing more than one type of class in the Collections

- The wildcard argument is specified by the '?' and it represents an unknown type

Wildcards generics

- The wildcard simply matches the validity of object

```
boolean same_Avg(Stats<?> ob){
        if(average() ==ob.average())
                return true;
        return false;
}
```

Generic method

- It is possible to declare a generic method that uses one or more type parameters
- Methods inside a generic class are automatically generic relative to the type parameters
- It is possible to create a generic method that is enclosed within a non-generic class.

Generic method

- The type parameters are declared before the return type of the method
- Generic methods can be either static or non-static

```
<type-param-list> ret-type method-name(param-list)
{…..}
```

Generic Interfaces

- Generic interfaces are specified just like generic classes.

```
interface MinMax<T extends Comparable<T>>
{       T min();
        T max();}
```

Generic Interfaces

```
interface MinMax<T extends Comparable<T>>
{       T min();
        T max();}

class Myclass<T extends Comparable<T>> implements
MinMax<T>
{
…….
}
```