



Generic

Session 10

Objectives

Identify the need for Generics

List the advantages and limitations of Generics

Explain generic class declaration and instantiation

Define and describe generic methods

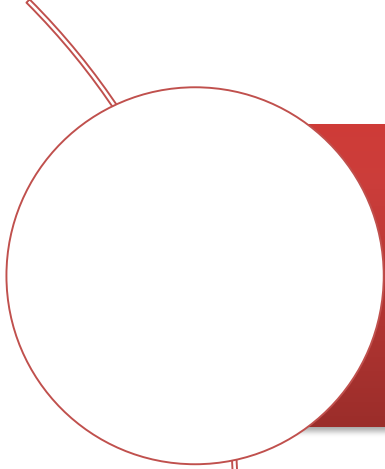
Describe the relationship between Collection and Generics

Explain the wildcard argument

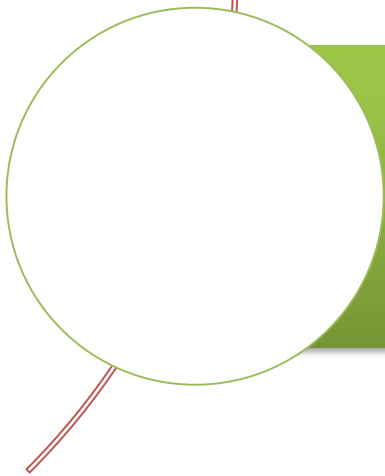
Describe the use of inheritance with Generics

Describe the use of legacy code in Generics and Generics in legacy code

Explain type inference



Genericity is a way by which programmers can specify the type of objects that a class can work with via parameters passed at declaration time and evaluated at compile time.



Generic types can be compared with functions which are parameterized by type variables and can be instantiated with different type arguments depending on the context.

- Generics in Java code generates one compiled version of a generic class.
- The introduction of Generics in Java classes will help remove the explicit casting of a class object so the ClassCastException will not arise during compilation.
- Generics will help to remove type inconsistencies during compile time rather than at run time.
- Generics are added to the Java programming language because they enable:
 - Getting more information about a collection's type.
 - Keeping track of the type of elements a collection contains.
 - Using casts all over the program.

- Generics allow the programmer to communicate the type of a collection to the compiler so that it can be checked.
- Thus, using Generics is safe as during compilation of the program, the compiler consistently checks for the element type of the collection and inserts the correct cast on elements being taken out of the collection.

Code Snippet

```
LinkedList list = new LinkedList();  
list.add(new Integer(1));  
Integer num = (Integer) list.get(0);
```

Code Snippet

```
LinkedList<Integer> list = new LinkedList<Integer>();  
list.add(new Integer(1));  
Integer num = list.get(0);
```

Advantages of Generics



Generics allow flexibility of dynamic binding.

Generic type helps the compiler to check for type correctness of the program at the compile time.

In Generics, the compiler detected errors are less time consuming to fix than runtime errors.

The code reviews are simpler in Generics as the ambiguity is less between containers.

In Generics, codes contain lesser casts and thus help to improve readability and robustness.

Limitations of Generics

In Generics, you cannot create generic constructors.



A local variable cannot be declared where the key and value types are different from each other.



Generic Classes



A generic class is a mechanism to specify the type relationship between a component type and its object type.

The syntax for declaring a generic class is same as ordinary class except that in angle brackets (<>) the type parameters are declared.

The declaration of the type parameters follows the class name.

The type parameters are like variables and can have the value as a class type, interface type, or any other type variable except primitive data type.

The class declaration such as `List<E>` denotes a class of generic type.

Generic Classes



The parameter to the generic class (`INTEGER` in an `ARRAY [INTEGER]`) is the class given in the array declaration and is bound at compile time.

A generic class can thus generate many types, one for each type of parameter, such as `ARRAY [TREE]`, `ARRAY [STRING]`, and so on.

Generic classes can accept one or more type parameters.

Therefore, they are called parameterized classes or parameterized types.

The type parameter section of a generic class can include several type parameters separated by commas.

Declare and Instantiate Generic Class

- To create an instance of the generic class, the new keyword is used along with the class name except that the type parameter argument is passed between the class name and the parentheses.
- The type parameter argument is replaced with the actual type when an object is created from a class.
- A generic class is shared among all its instances.

Syntax

```
class NumberList <Element> {...}
```

Declare and Instantiate Generic Class

- The code creates a generic type class declaration with a type variable, T that can be used anywhere in the class.
- To refer to this generic class, a generic type invocation is performed which replaces T with a value such as String.
- Typically, type parameter names are single, uppercase letters.
- Following are the commonly used type parameter names:
 - K - Key
 - T - Type
 - V - Value
 - N - Number
 - E - Element
 - S, U, V, and so on

■ Generic methods

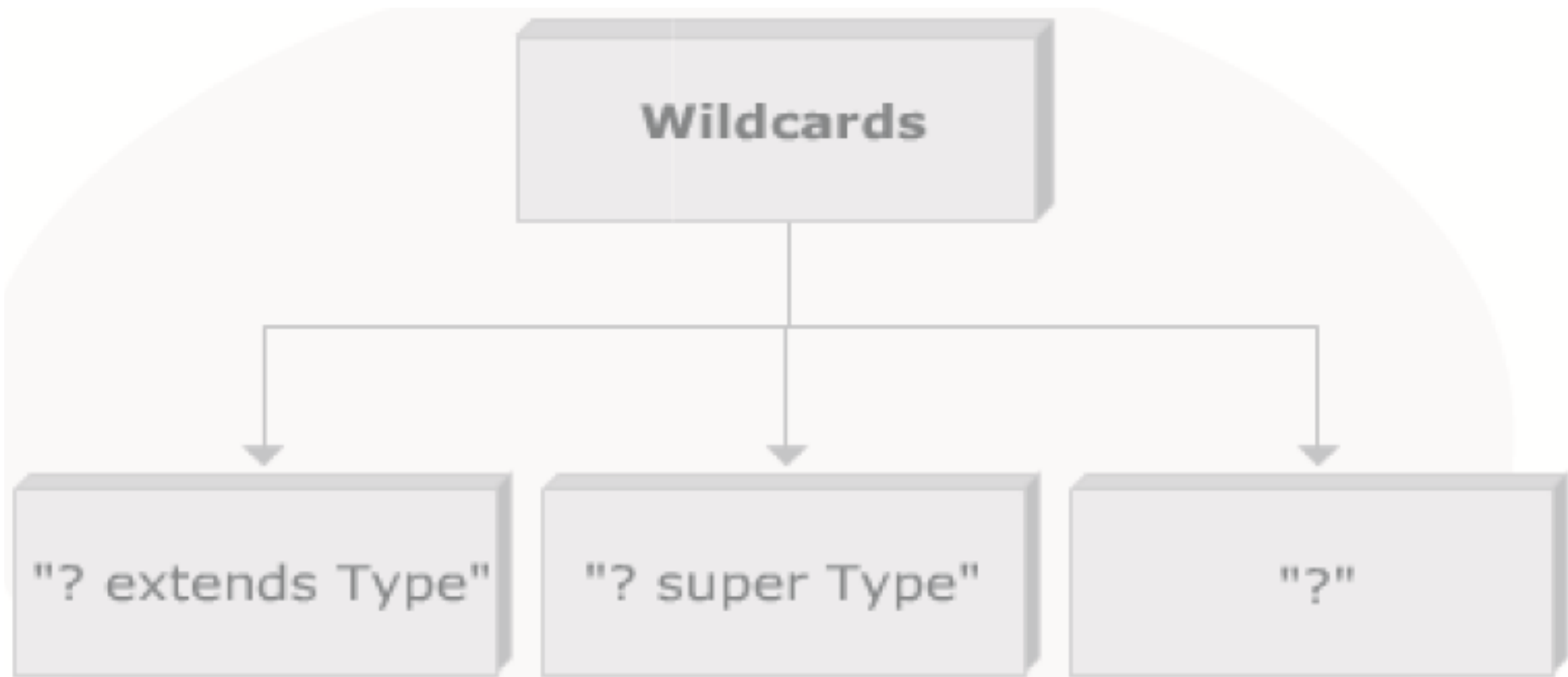
- Are defined for a particular method and have the same functionality as the type parameter has for generic classes.
 - Can appear in generic classes as well as in nongeneric classes.
 - Can be defined as a method with type parameters.
 - Are best suited for overloaded methods that perform identical operations for different argument type.
 - Make the overloaded methods more compact and easy to code.
- A generic method allows type parameters used to make dependencies among the type of arguments to a method and its return type.
 - The return type does not depend on the type parameter, or any other argument of the method.
 - This shows that the type argument is being used for polymorphism

Code Snippet

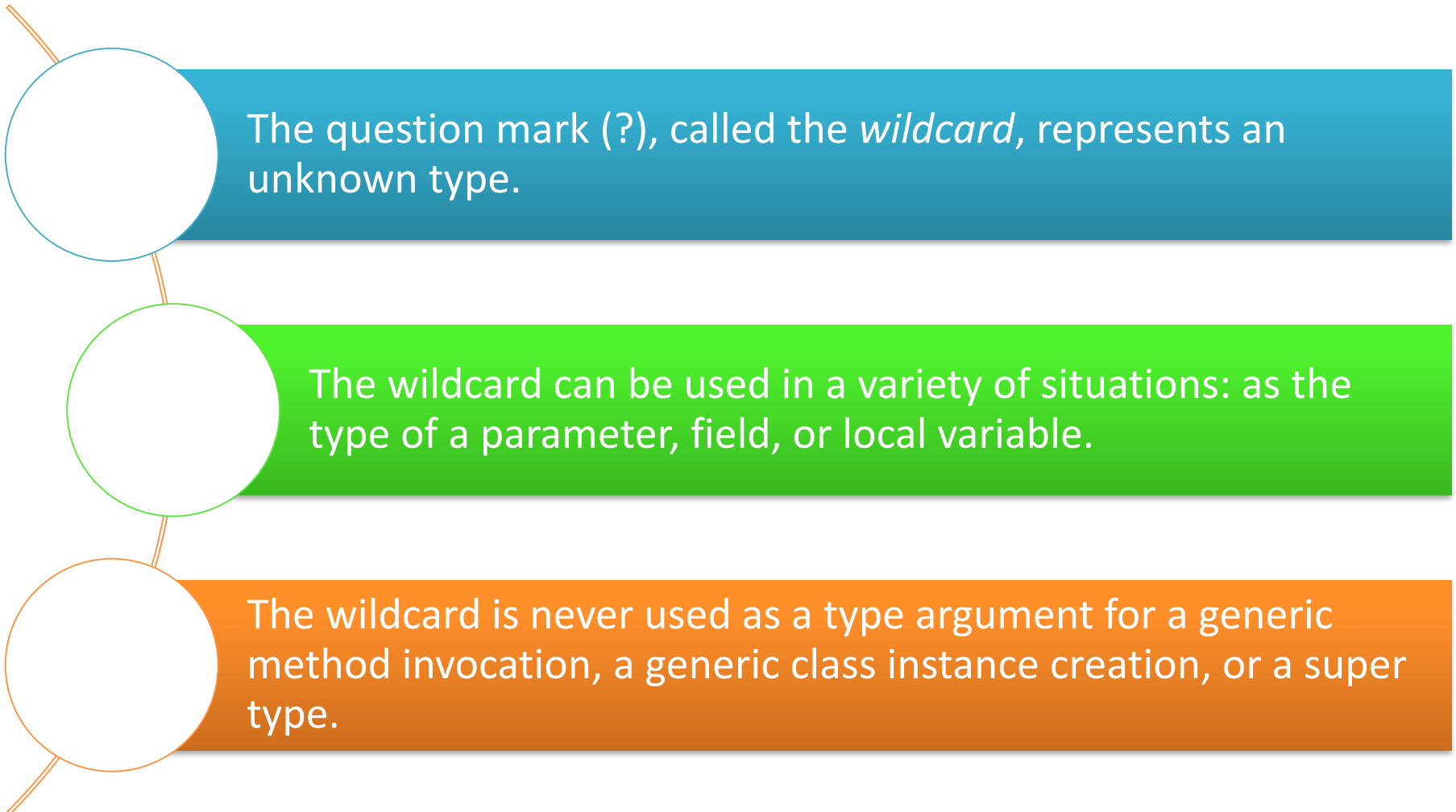
```
class NumberList <T>
{
    public<T> void display(T[] val)
    {
        for( T element : val)
        {
            System.out.printf("Values are: %s " , element);
        }
    }
}
```

- A single generic method declaration can be called with arguments of different types.
- Generic methods can be defined based on the following rules:
 - Each type parameter section includes one or more type parameters separated by commas. A type parameter is an identifier that specifies a generic type name.
 - All generic method declarations have a type parameter section delimited by angle brackets preceding the method's return type.
 - A generic method's body should include type parameters that represent only reference types.
 - The type parameters can be used to declare the return type. They are placeholders for the types of the arguments passed to the generic method. These arguments are called actual type arguments.

Wildcards with Generics



Wildcards with Generics



The question mark (?), called the *wildcard*, represents an unknown type.

The wildcard can be used in a variety of situations: as the type of a parameter, field, or local variable.

The wildcard is never used as a type argument for a generic method invocation, a generic class instance creation, or a super type.

Upper Bounded Wildcards

- Use an upper bounded wildcard to relax the restrictions on a variable.

```
public static void process(List<? extends Foo> list) { ..  
}
```

- The upper bounded wildcard, `<? extends Foo>`, where `Foo` is any type, matches `Foo` and any subtype of `Foo`.

- A *lower bounded* wildcard restricts the unknown type to be a specific type or a *super type* of that type.
- Syntax to define lower bounded wildcard
 <? super className>
- *Sample: the method that works on lists of Integer and the super types of Integer, such as Integer, Number, and Object*

```
public static void addNumbers(List<? super Integer> list)
{
    for (int i = 1; i <= 10; i++) { list.add(i); }
}
```

- A *raw type* is the name of a generic class or interface without any type arguments.

```
public class Box<T> {
    public void set(T t) { /* ... */ }
    // ...
}
//...
Box<String> stringBox = new Box<>();
Box rawBox = stringBox;
Box rawBox = new Box();      // rawBox is a raw type of Box<T>
Box<Integer> intBox = rawBox; // warning: unchecked conversion
```

- Generics in Java code generate one compiled version of a generic class.
- Generics help to remove type inconsistencies during compile time rather than at run time.
- There are three types of wildcards used with Generics such as ‘? extends Type’, ‘? super Type’, and ‘?’.
- Generic methods are defined for a particular method and have the same functionality as the type parameter have for generic classes.
- Type parameters are declared within the method and constructor signature when creating generic methods and constructors.
- A single generic method declaration can be called with arguments of different types.

ĐẠI HỌC FPT CẦN THƠ

