CSE3040 Java Language

Lecture 15: Generic Programming (2)

Dept. of Computer Engineering,

Sogang University

Generic Classes: Cases

- A generic class can extend a non-generic class

```
class Shape { }
class FruitBox<T> extends Shape { }
```

- A generic class can extend a generic class, using the same type parameter.

```
class Box<T> {
    ArrayList<T> list = new ArrayList<T>();
    void add(T item) { list.add(item); }
    T get(int i) { return list.get(i); }
    int size() { return list.size(); }
    public String toString() { return list.toString(); }
}

class FruitBox<T> extends Box<T> { }
```

- An inheriting generic class can further limit the type.

```
class Box<T> { ... }
class FruitBox<T extends Fruit> extends Box<T> { }
```

서강대학교
SOGANG UNIVERSITY

Generic Classes: Cases

- If a superclass limits types, then its subclass should also limit types.

```
class Box<T extends Fruit> { ... }
class FruitBox<T> extends Box<T> { ... }    // Error
```

```
class Box<T extends Fruit> { ... }
class FruitBox<T extends Fruit> extends Box<T> { ... }    // OK
```

Generic Classes: Wild Card

- Class Juicer has a static method that gets a FruitBox and returns Juice.
  - Juicer is not a generic class, so cannot use a type variable in the method makeJuice.

```
class Juicer {
    static Juice makeJuice(FruitBox<Fruit> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
```

- Since the parameter is of type FruitBox<Fruit>, only an instance of FruitBox<Fruit> can be passed as a parameter to this method.

```
FruitBox<Fruit> fruitBox = new FruitBox<Fruit>();
FruitBox<Apple> appleBox = new FruitBox<Apple>();
fruitBox = new FruitBox<Fruit>();
appleBox = new FruitBox<Apple>();
System.out.println(Juicer.makeJuice(fruitBox));    // OK
System.out.println(Juicer.makeJuice(appleBox));    // Error: the argument is not of type FruitBox<Fruit>.
```

DETOUR: for-each statements

- Java supports a special type of for loop for iterable objects.
  - Iterable objects: arrays, ArrayList, etc.

```
String[] numbers = {"one", "two", "three"};
for(int i=0; i<numbers.length; i++) {
    System.out.println(numbers[i]);
}
```

```
String[] numbers = {"one", "two", "three"};
for(String number: numbers) {
    System.out.println(number);
}
```

  - After each iteration, the next item in the object numbers is assigned to variable number, and the code block is executed.

Generic Classes: Wild Card

- In order to make a method makeJuice for FruitBox<Apple>, we need to make a separate method.
- However, this creates a compiler error.
  - A generic class with different parameterized type is not allowed.

```
class Juicer {
    static Juice makeJuice(FruitBox<Fruit> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
    static Juice makeJuice(FruitBox<Apple> box) {        // Error: another method exists
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
```

Generic Classes: Wild Card

- Instead, in order to support multiple types of classes, we use the wild card '?'.
  - <? extends T>: class T and all its descendants.
  - <? super T>: class T and all its ascendants.
  - <?>: all types. Same as <? extends Object>

```
class Juicer {
    static Juice makeJuice(FruitBox<? extends Fruit> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
```

```
FruitBox<Fruit> fruitBox = new FruitBox<Fruit>();
FruitBox<Apple> appleBox = new FruitBox<Apple>();
fruitBox = new FruitBox<Fruit>();
appleBox = new FruitBox<Apple>();
System.out.println(Juicer.makeJuice(fruitBox));    // OK
System.out.println(Juicer.makeJuice(appleBox));    // OK!
```

Generic Classes: Example 3

```
import java.util.ArrayList;
class Fruit { public String toString() { return "Fruit"; } }
class Apple extends Fruit { public String toString() { return "Apple"; } }
class Grape extends Fruit { public String toString() { return "Grape"; } }
class Juice {
    String name;
    Juice(String name) { this.name = name + "Juice"; }
    public String toString() { return name; }
}
class Juicer {
    static Juice makeJuice(FruitBox<? extends Fruit> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
class Box<T> {
    ArrayList<T> list = new ArrayList<T>();
    void add(T item) { list.add(item); }
    T get(int i) { return list.get(i); }
    ArrayList<T> getList() { return list; }
    int size() { return list.size(); }
    public String toString() { return list.toString(); }
}
class FruitBox<T extends Fruit> extends Box<T> { }
```

Generic Classes: Example 3

```
public class Lecture {
    public static void main(String[] args) {
        FruitBox<Fruit> fruitBox = new FruitBox<Fruit>();
        FruitBox<Apple> appleBox = new FruitBox<Apple>();

        fruitBox.add(new Apple());
        fruitBox.add(new Grape());
        appleBox.add(new Apple());
        appleBox.add(new Apple());

        System.out.println(Juicer.makeJuice(fruitBox));
        System.out.println(Juicer.makeJuice(appleBox));
    }
}
```

Generic Methods

- We can define a generic method by inserting type variable before the return type.

```
class Juicer {
    static <T extends Fruit> Juice makeJuice(FruitBox<T> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
```

- In order to call a generic method, we add the type variable before the method name.

```
FruitBox<Fruit> fruitBox = new FruitBox<Fruit>();
FruitBox<Apple> appleBox = new FruitBox<Apple>();
fruitBox = new FruitBox<Fruit>();
appleBox = new FruitBox<Apple>();
System.out.println(Juicer.<Fruit>makeJuice(fruitBox));
System.out.println(Juicer.<Apple>makeJuice(appleBox));
```

- If the compiler can figure out the type, then the parameterized type can be omitted.

```
System.out.println(Juicer.makeJuice(fruitBox));
System.out.println(Juicer.makeJuice(appleBox));
```

Casting Generic Types

- Type casting between generic classes with the same raw type and different parameterized types is not allowed.

```
Box<Object> objBox = null;
Box<String> strBox = null;
objBox = (Box<Object>)strBox;    // Error: cannot cast from Box<String> to Box<Object>
strBox = (Box<String>)objBox;    // Error: cannot cast from Box<Object> to Box<String>
```

- If the parameterized type uses the wild card and thus includes class String, then casting is allowed.

```
Box<? extends Object> objBox = new Box<String>();    // OK
```

Casting Generic Types

- We can use wild card to allow multiple types.

```
class Box<T> {
    ArrayList<T> list = new ArrayList<T>();
    void add(T item) { list.add(item); }
    T get(int i) { return list.get(i); }
    ArrayList<T> getList() { return list; }
    int size() { return list.size(); }
    public String toString() { return list.toString(); }
}
class FruitBox<T extends Fruit> extends Box<T> { }
```

  - we choose T as "? extends Fruit", which allows all descendants of class Fruit.
  - then, we assign an instance of class FruitBox<Fruit> to a FruitBox<? extends Fruit> type variable.

```
FruitBox<? extends Fruit> box = new FruitBox<Fruit>();    // OK
System.out.println(box);
```

```
FruitBox<? extends Fruit> box = new FruitBox<Apple>();    // OK
System.out.println(box);
```

Casting Generic Types

- If we go backwards, the compiler does not create an error but creates a warning.
  - warning: unchecked cast from FruitBox<? extends Fruit> to FruitBox<Apple>.

```
FruitBox<? extends Fruit> box = new FruitBox<Fruit>();
FruitBox<Apple> appleBox = (FruitBox<Apple>)box;
```

Erasure of Generic Type

- The Java compiler checks the source files with generic types and modifies the source code so that generic types are removed.
- Thus, generic types do not exist in object (.class) files.

- The compiler first converts the type variable to a particular class.

```
class Box<T> {
    void add(T t) {
        ...
    }
}
```

→

```
class Box {
    void add(Object t) {
        ...
    }
}
```

```
class FruitBox<T extends Fruit> {
    void add(T t) {
        ...
    }
}
```

→

```
class FruitBox {
    void add(Fruit t) {
        ...
    }
}
```

Erasure of Generic Type

- The compiler first converts the type variable to a particular class.

```
public class Entry<K, V> {
    private K key;
    private V value;
    public Entry(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public K getKey() { return key; }
    public V getValue() { return value; }
}
```

⇩

```
public class Entry {
    private Object key;
    private Object value;
    public Entry(Object key, Object value) {
        this.key = key;
        this.value = value;
    }
    public Object getKey() { return key; }
    public Object getValue() { return value; }
}
```

Erasure of Generic Type

- After removing the generic type, type casts are added if necessary.

```java
T get(int i) {
    return list.get(i);
}
```

```java
Fruit get(int i) {
    return (Fruit)list.get(i);
}
```

```java
Entry<String, Integer> entry = ...;
String key = entry.getKey();
```

```java
Entry<String, Integer> entry = ...;
String key = (String)entry.getKey();
```

Programming Lab #15

서강대학교
SOGANG UNIVERSITY

15-01. Generic Classes and Generic Methods

- Write the following code and understand the results.
- Try modifying method makeJuice to a generic method.

```java
import java.util.ArrayList;
class Fruit { public String toString() { return "Fruit"; } }
class Apple extends Fruit { public String toString() { return "Apple"; } }
class Grape extends Fruit { public String toString() { return "Grape"; } }
class Juice {
    String name;
    Juice(String name) { this.name = name + "Juice"; }
    public String toString() { return name; }
}
class Juicer {
    static Juice makeJuice(FruitBox<? extends Fruit> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return new Juice(tmp);
    }
}
class Box<T> {
    ArrayList<T> list = new ArrayList<T>();
    void add(T item) { list.add(item); }
    T get(int i) { return list.get(i); }
    ArrayList<T> getList() { return list; }
    int size() { return list.size(); }
    public String toString() { return list.toString(); }
}
class FruitBox<T extends Fruit> extends Box<T> { }
```

15-01. Generic Classes and Generic Methods

```
public class Lecture {
    public static void main(String[] args) {
        FruitBox<Fruit> fruitBox = new FruitBox<Fruit>();
        FruitBox<Apple> appleBox = new FruitBox<Apple>();

        fruitBox.add(new Apple());
        fruitBox.add(new Grape());
        appleBox.add(new Apple());
        appleBox.add(new Apple());

        System.out.println(Juicer.makeJuice(fruitBox));
        System.out.println(Juicer.makeJuice(appleBox));
    }
}
```

End of Class

Instructor office: AS818A

Email: jso1@sogang.ac.kr