Lecture 22:Generics

CS 0445: Data Structures

Constantinos Costa

http://db.cs.pitt.edu/courses/cs0445/current.term/

Oct 31, 2019, 8:00-9:15
University of Pittsburgh, Pittsburgh, PA

The Interface `Comparable`

- Consider the method **`compareTo`** for class **`String`**
- if **`s`** and **`t`** are strings, **`s.compareTo(t)`** is
  - Negative if **`s`** comes before **`t`**
  - Zero if **`s`** and **`t`** are equal
  - Positive if **`s`** comes after **`t`**

The Interface `Comparable`

- By invoking **`compareTo`**, you compare two objects of the class T.
- LISTING JI5-3 The interface **java.lang.Comparable**

```
public interface Comparable<T>
{
   public int compareTo(T other);
} // end Comparable
```

The Interface `Comparable`

- Create a class **`Circle`**, define **`compareTo`**

```
public class Circle implements Comparable<Circle>, Measurable
{
  private double radius;

  // Definitions of constructors and methods are here.
  // . . .

  public int compareTo(Circle other)
  {
    int result;
    if (this.equals(other))
      result = 0;
    else if (radius < other.radius)
      result = -1;
    else
      result = 1;

    return result;
  } // compareTo
} // end Circle
```

Generic Methods

- An example of a generic method

```
public class Example
{
  public static <T> void displayArray(T[] anArray)
  {
    for (T arrayEntry : anArray)
    {
      System.out.print(arrayEntry);
      System.out.print(' ');
    } // end for
    System.out.println();
  } // end displayArray

  public static void main(String args[])
  {
    String[] stringArray = {"apple", "banana", "carrot", "dandelion"};
    System.out.print("stringArray contains ");
    displayArray(stringArray);

    Character[] characterArray = {'a', 'b', 'c', 'd'};
    System.out.print("characterArray contains ");
    displayArray(characterArray);
  } // end main
} // end Example
```

Bounded Type Parameters

- Consider this simple class of squares:

```
public class Square<T>
{
  private T side;

  public Square(T initialSide)
  {
    side = initialSide;
  } // end constructor

  public T getSide()
  {
    return side;
  } // end getSide
} // end Square
```

Different types of square objects possible.

```
Square<Integer> intSquare = new Square<>(5);
Square<Double> realSquare = new Square<>(2.1);
Square<String> stringSquare= new Square<>("25");
```

Bounded Type Parameters

- Imagine that we want to write a static method that returns the smallest object in an array.
  Suppose that we wrote our method shown here:

```
public MyClass
{
  // First draft and INCORRECT:
  public static <T> T arrayMinimum(T[] anArray)
  {
    T minimum = anArray[0];
    for (T arrayEntry : anArray)
    {
      if (arrayEntry.compareTo(minimum) < 0)
        minimum = arrayEntry;
    } // end for

    return minimum;
  } // end arrayMinimum
} // end MyClass
```

Bounded Type Parameters

- Header really should be as shown

```
public MyClass
{
  public static <T extends Comparable<T>> T arrayMinimum(T[] anArray)
  {
    T minimum = anArray[0];
    for (T arrayEntry : anArray)
    {
      if (arrayEntry.compareTo(minimum) < 0)
        minimum = arrayEntry;
    } // end for

    return minimum;
  } // end arrayMinimum
  // . . .
} // end MyClass
```

Wildcards

- Question mark, ?, is used to represent an unknown class type
  - Referred to as a wildcard
- Consider following method and objects

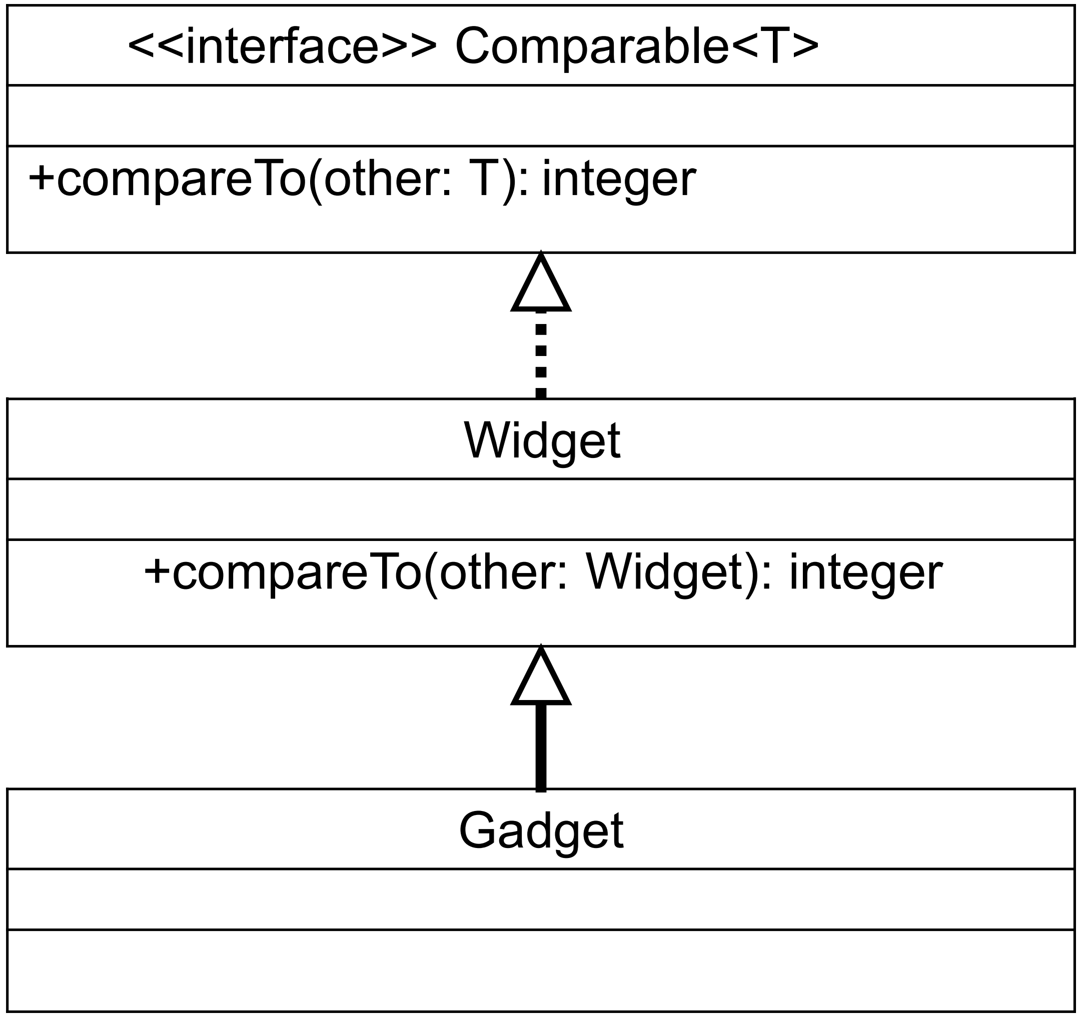
```
 public static void displayPair(OrderedPair<?> pair)
 {
   System.out.println(pair);
 } // end displayPair
OrderedPair<String> aPair = new OrderedPair<>("apple", "banana");
OrderedPair<Integer> anotherPair = new OrderedPair<>(1, 2);
```

- **Method `displayPair` will accept as an argument a pair of objects whose data type is any one class**

```
displayPair(aPair);
displayPair(anotherPair);
```

Bounded Wildcards

- The class `Gadget` is derived from the class `Widget`, which implements the interface `Comparable`



<<interface>> Comparable<T>
+compareTo(other: T): integer
Widget
+compareTo(other: Widget): integer
Gadget

Happy **Halloween**!