University of Glasgow

13 Generic abstraction

- Genericity
- Generic classes
- Generic procedures

University of Glasgow

Genericity

- A program-unit is **generic** if it is parameterized with respect to a type on which it depends.
- Many reusable program-units (e.g., stack, queue, list, and set ADTs/classes) are naturally generic.
- Generic program-units include:
  - generic packages (not covered here)
  - generic classes
  - generic procedures.

University of Glasgow

Generic classes

- In Java, a **generic class** *GC* is parameterized with respect to a type *T* on which it depends:

```
class GC <T> {
    … T … T …
}
```

- The generic class must be **instantiated**, by substituting a type argument *A* for the type parameter *T*:

```
GC<A>
```

- This instantiation generates an ordinary class.

University of Glasgow

Example: Java generic class *(1)*

- Consider a class that encapsulates lists with elements of type `T`. This class can be made generic with respect to type `T`.
- Generic class declaration:

type parameter

```
class List <T> {
    // A List<T> object is a list of elements of type
    // T, where the number of elements ≤ cap.

    private static final cap = 100;
    private int size;
    private T[] elems;
```

University of Glasgow

Example: Java generic class *(2)*

- Generic class declaration *(continued)*:

```
    public List () {
    // Construct an empty list.
        size = 0;
        elems = (T[]) new Object[cap];
    }

    public void add (T elem) {
    // Add elem to the end of this list.
        elems[size++] = elem;
    }

}
```

University of Glasgow

Example: Java generic class *(3)*

- The following instantiation generates a class that encapsulates lists with `String` elements:

```
List<String>
```

type argument (substituted for type parameter `T`)

- The generated class can be used like an ordinary class:

```
List<String> sentence;
sentence = new List<String>();
sentence.add("…");
```

University of Glasgow

Example: Java generic class *(4)*

- The following instantiation generates a class that encapsulates lists with `Date` elements:

```
List<Date> holidays;
holidays = new List<Date>();
holidays.add (new Date(2009, 1, 1));
```

- In an instantiation, the type argument must be a class, *not a primitive type*:

```
List<int> primes;
```

illegal

University of Glasgow

Java generic interface `Comparable<T>`

- Java also supports generic interfaces.
- From `java.lang`:

```
interface Comparable <T> {
    public int compareTo (T that);
}
```

- If class `C` is declared as implementing `Comparable<C>`, `C` *must* be equipped with a `compareTo` method that compares objects of type `C`.

University of Glasgow

Bounded type parameters *(1)*

- Consider a generic class *GC‹T›* that requires *T* to be equipped with some specific methods.
- *T* may be specified as **bounded** by a class *C*:

```
class GC <T extends C> { … }
```

  - *T* is known to be equipped with all the methods of *C*.
  - The type argument must be a subclass of *C*.
- Alternatively, *T* may be specified as **bounded** by an interface *I*:

```
class GC <T extends I > { … }
```

  - *T* is known to be equipped with all the methods of *I*.
  - The type argument must be a class that implements *I*.

University of Glasgow

Bounded type parameters *(2)*

- Recall a generic class *GC‹T›* that does *not* require *T* to be equipped with any specific methods. As we have seen, it is enough just to name *T*:

```
class GC <T > { ... }
```

- This is actually an abbreviation for:

```
class GC <T  extends Object> { ... }
```

  - *T* is known to be equipped with all the `Object` methods, such as `equals`.
  - The type argument must be a subclass of `Object`, i.e., any class.

University of Glasgow

Example: Java generic class with bounded type parameter *(1)*

- Consider a class `Pqueue<T>` that encapsulates priority queues with elements of type `T`. It is required that `T` is equipped with a `compareTo` method.
- Generic class declaration:

```
class PQueue <T extends Comparable<T>> {

    private static final cap = 20;
    private int size;
    private T[] elems;

    public PQueue () {
        size = 0;
        elems = (T[]) new Object[cap];
    }
```

Example: Java generic class with bounded type parameter *(2)*

- Generic class declaration *(continued)*:

```
    public void add (T elem) {
    // Add elem to this priority queue.
        if (elem.compareTo(elems[0])<0) …
        …
    }

    public T first () {
    // Return the first element of this priority queue.
        return elems[0];
    }

}
```

University of Glasgow

Example: Java generic class with bounded type parameter *(3)*

- Class `String` implements `Comparable<String>`. So the following generates a class that encapsulates priority queues with `String` elements:

```
PQueue<String> pq;
pq = new PQueue<String>();
pq.add("beta");
pq.add("alpha");
out.print(pq.first());
```

Example: Java generic class with bounded type parameter *(4)*

- Suppose that class `Date` implements `Comparable<Date>`. Then the following generates a class that encapsulates priority queues with `Date` elements:

```
PQueue<Date> holidays;
```

- But `PQueue` *cannot* be instantiated with a class `C` that does not implement `Comparable<C>`:

```
PQueue<Button> buttons;
```

illegal

University of Glasgow

Generic procedures

- A Java method may be parameterized with respect to a type *T* on which the method depends.

University of Glasgow

Example: Java generic method

- A method that chooses between two arguments of type `T` can be made generic w.r.t. type `T`:

```
public static <T>
        T either (boolean b, T y, T z) {
    return (b ? y : z);
}
```

- Calls:

```
... either (isletter(c), c, '*')
... either (m > n, m, n)
```

implicitly substituting type `Character` for `T`

implicitly substituting type `Integer` for `T`