CISC 3115 TY3

C23b: Generics: Raw Type and Wildcards

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Discussed
  - Motivation of generics
  - Define generic classes and methods
  - Bounded generic type
- To discuss
  - Raw types
  - Wildcards
  - Erasures and restrictions

Raw type

- A generic class or interface used without specifying a concrete type for a type parameter.
- What is it for?
  - Generics was introduced to Java since JDK 1.5. How about the code written before then?
  - Raw types provides backward compatibility

Raw Type: Example

```
// raw type
ArrayList list = new ArrayList();

Comparable o1 = new ComparableRectangle();
```

Raw Type is Unsafe

- Why? See the following example. Does it compile? What happens when we run it?

```
// Max.java: Find a maximum object
public class Max {
  /** Return the maximum between two objects */
  public static Comparable max(Comparable o1, Comparable o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

Raw Type is Unsafe

- What happens?

```
Max.max("Welcome", 23);
```

Can We Make it Safe?

- Revise it to use a concrete type:

```
// Max1.java: Find a maximum object
public class Max1 {
  /** Return the maximum between two objects */
  public static <E extends Comparable<E>> E max(E o1, E o2) {
    if (o1.compareTo(o2) > 0)
      return o1;
    else
      return o2;
  }
}
```

Make it Safe

- How about now? What happens when we compile the code?

```
Max.max("Welcome", 23);
```

Best Practice: Avoiding Unsafe Raw Types

- Use

```
new ArrayList<ConcreteType>()
```

instead of

```
new ArrayList();
```

Questions?

- Raw type? What is it?
- Should we use raw types? What is the recommended practice?

Wildcards

- Why wildcards are necessary? See this example.

  ? unbounded wildcard

  - Equivalent to ? extends Object
  - Object or a subtype of Object

  ? extends T bounded wildcard

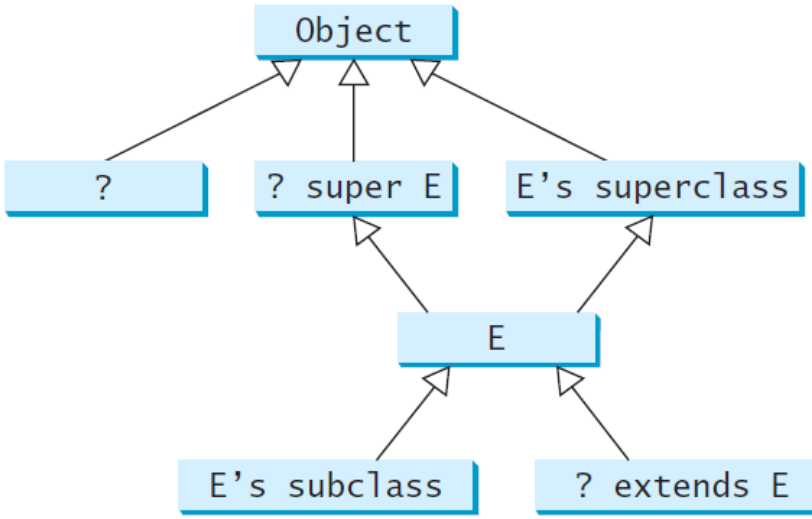
  - T or a subtype of T

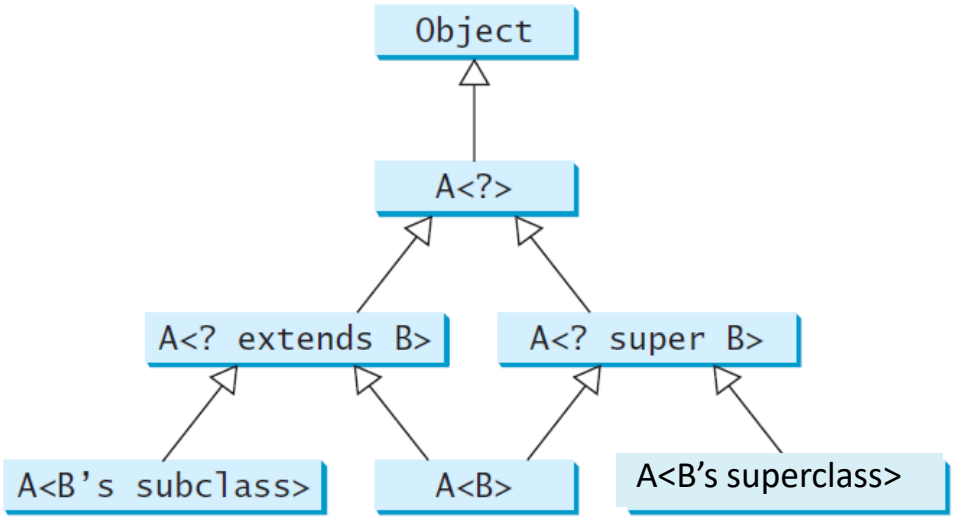
  ? super T lower bound wildcard

  - T or a supertype of T

Generic Types and Wildcard Types

- They forms a hierarchy: A and B are data types (classes or interfaces), E is a generic type





Questions?

- Concept of wildcard types
- Have you seen them?

Erasure and Restrictions on Generics

- Generics are implemented using an approach called type erasure, that is,
  - The compiler uses the generic type information to compile the code, but erases it afterwards.
  - So the generic information is not available at run time.
  - This approach enables the generic code to be backward-compatible with the legacy code that uses raw types.

Generics: Compile Time Checking

- Example
  - The compiler checks whether generics is used correctly for the following code
  - And translates it into the equivalent code below for runtime use. The code uses the raw type.

```
ArrayList<String> list = new ArrayList<>();
list.add("Oklahoma");
String state = list.get(0);
```

(a)

```
ArrayList list = new ArrayList();
list.add("Oklahoma");
String state = (String)(list.get(0));
```

(b)

Important Facts

- A generic class is shared by all its instances regardless of its actual generic type.

```
GenericStack<String> stack1 = new GenericStack<>();

GenericStack<Integer> stack2 = new GenericStack<>();
```

- Although GenericStack<String> and GenericStack<Integer> are two types, but there is only one class GenericStack loaded into the JVM

Restrictions on Generics

- Restriction 1: Cannot Create an Instance of a Generic Type. (i.e., new E()).
- Restriction 2: Generic Array Creation is Not Allowed. (i.e., new E[100]).
- Restriction 3: A Generic Type Parameter of a Class Is Not Allowed in a Static Context.
- Restriction 4: Exception Classes Cannot be Generic.

Questions?

- How does the Java compiler deal with generic types?
- What are the restrictions?