CISC 3115 TY3

# C23a: Generics: Motivation and Definition

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

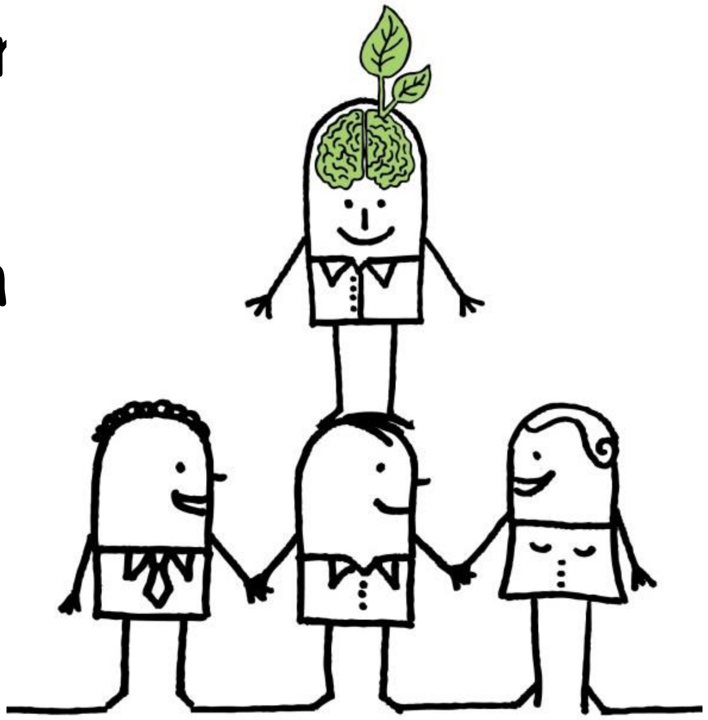
## Outline

- Motivation of generics
- Define generic classes and methods
- Bounded generic type

# Reuse, Reuse, and Reuse …

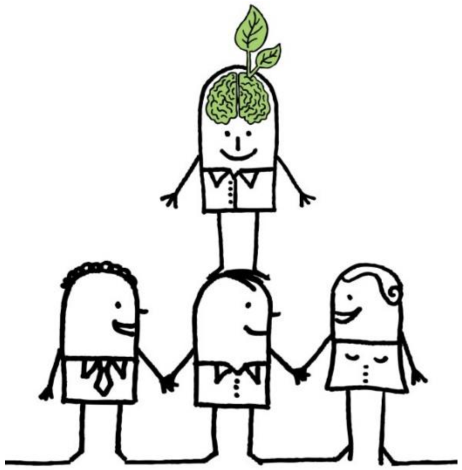
"If I have seen further it is by standing on the shoulder of Giants."

-- Isaac Newton



# Generics: Motivation

- To write code that can be applied to many data types
  - e.g., if the algorithms is essentially the same, why should we write a second time?
  - But, how do we do it?
    - Solution 1: use type hierarchy
    - Solution 2: use "generics"
- To detect errors at compilation time other than at runtime by introducing "generics"



## Solution 1: Use Type Hierarchy

- Consider the design of the Comparable interface

```
public interface Comparable {
    public int compareTo(Object o);
}
```

- What's the problem?

## Solution 2: Use “Generics”

- Consider the design of the Comparable interface

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

  - where “T” represents a formal generic type, which can be replaced later with an actual concrete type.

## Generics

- *Generics* is the capability to parameterize data types.
- Generic instantiation: with this capability, one use generic types when defining a class or a method, and the generic types can be substituted using concrete types by the compiler.
- Be aware that “concrete” has a different meaning in the context of generics from “concrete” in concrete subtype in the context of inheritance.

## Generic Instantiation: Example

- Consider the design of the Comparable interface

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

- And an implementation of Comparable<T>

```
public class Shape implements Comparable<Shape> {
    public int compareTo(Shape s);
}
```

where “T” is replaced by a concrete type “Shape”.

## Solution 2: Use "Generics"

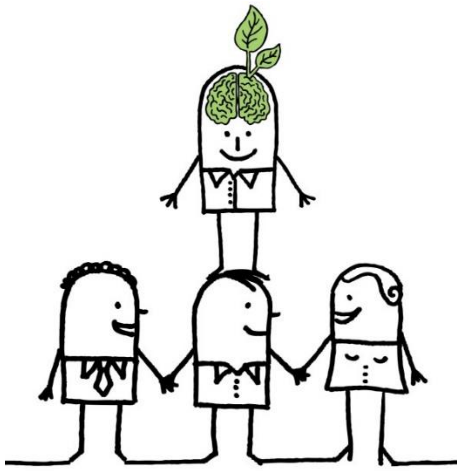
- Consider the design of the Comparable interface

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

  - where "T" represents a formal generic type, which can be replaced later with an actual concrete type.
- <u>What's the benefit?</u>

# Generics: Benefits

- To write code that can be applied to many data types
  - e.g., if the algorithms is essentially the same, why should we write a second time?
  - But, how do we do it?
    - Solution 1: use type hierarchy
    - <u>Solution 2: use "generics"</u>
- To detect errors at compilation time other than at runtime by <u>introducing "generics"</u>



## Detecting Errors

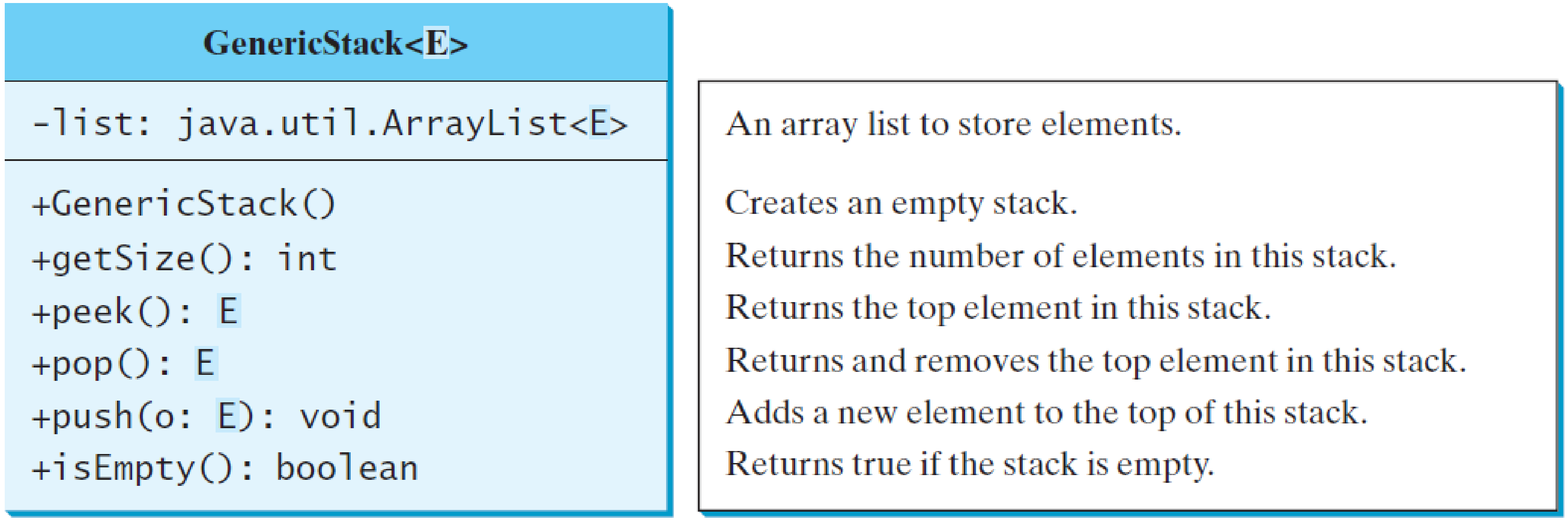
- A generic class or method permits one to specify allowable types of objects that the class or method may work with.
- If one attempts to use the class or method with an incompatible object, a compilation error occurs.

## Questions?

- Concept of *generic type* and *generic instantiation*
- Benefit of using *generic types*.

## Defining Generic Classes and Interfaces

- Example: using ArrayList to design and implement a generic Stack data structure



| GenericStack<E> | |
|---|---|
| -list: java.util.ArrayList<E> | An array list to store elements. |
| +GenericStack() | Creates an empty stack. |
| +getSize(): int | Returns the number of elements in this stack. |
| +peek(): E | Returns the top element in this stack. |
| +pop(): E | Returns and removes the top element in this stack. |
| +push(o: E): void | Adds a new element to the top of this stack. |
| +isEmpty(): boolean | Returns true if the stack is empty. |

## Generic Methods

- Instance methods
- Static methods

## Generic Methods

- Example using generic type “E”

```
public static <E> void print(E[] list) {
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
    System.out.println();
  }
```

- Compare it with the one using type hierarchy

```
public static void print(Object[] list) {
    for (int i = 0; i < list.length; i++)
      System.out.print(list[i] + " ");
    System.out.println();
  }
```

## Bounded Generic Type

- A generic type can be specified as a subtype of another type
- This generic type is then “bounded” (to a type)

# Bounded Generic Type: Example

```
public static <E extends GeometricObject> boolean
      equalArea(E object1, E object2) {
   return object1.getArea() == object2.getArea();
}



public static void main(String[] args ) {
   Rectangle rectangle = new Rectangle(2, 2);
   Circle circle = new Circle (2);
   System.out.println("Same area? " + equalArea(rectangle, circle));
}
```

# Questions?

- Define generic classes and methods
- Use generic classes and methods
  - Instance methods
  - Static methods
- Bounded generic types