Software Construction
Java Generics

Dhammika Elkaduwe

Department of Computer Engineering
Faculty of Engineering

University of Peradeniya

ILO: what to look for

- The *Object* key word and its applications
- Idea of generics
- Types as parameters

Generics: Motivation

```
void bubble_Sort(int [] data);
int getMid(int []); // find the mid element of array
```

- Bubble sort will work with other data types as well
- (sensible) reuse of code?
- Keep the type safety!

Generics: Motivation: Example
see BadLinkedList.java

Want to implement a linked list. Here is one **bad** way of doing it!

```
public class BadLinkedList {
    int nextItem;
    int max;
    Object [] data;
    private static final int blockSize = 100;

public BadLinkedList(int max) {
this.nextItem = 0;
this.max = blockSize;
data = new Object[blockSize];
    }
    ...........
}
```

Generics: Motivation: Example
see BadLinkedList.java

Want to implement a linked list. Here is one **bad** way of doing it!

```
public class BadLinkedList {
    int nextItem;
    int max;
    Object [] data;
    private static final int blockSize = 100;

public BadLinkedList(int max) {
this.nextItem = 0;
this.max = blockSize;
data = new Object[blockSize];
    }
    ...........
}
```

Generics: Motivation: Example
see BadLinkedList.java

Want to implement a linked list. Here is one **bad** way of doing it!

```
public class BadLinkedList {
    int nextItem;
    int max;
    Object [] data;
    private static final int blockSize = 100;

public BadLinkedList(int max) {
this.nextItem = 0;
this.max = blockSize;
data = new Object[blockSize];
    }
    ...........
}
```

Generics: Motivation: Example
see BadLinkedList.java

Want to implement a linked list. Here is one **bad** way of doing it!

```
private void more() {
  int size = this.max + blockSize; // add blockSize more
      elements
  Object [] newData = new Object[size];
  for(int i=0; i<this.max; i++)
        newData[i] = this.data[i];

  this.data = newData;
  this.max = size;
}

public void add(Object o) {
  if(isFull()) more();
  this.data[this.nextItem++] = o;
}
```

Generics: Motivation: Example
see BadLinkedList.java

Want to implement a linked list. Here is one **bad** way of doing it!

```
Object remove() {
  if(isEmpty()) return null;
  return this.data[--this.nextItem];
}
```

What do we have as BadLinkedList.java:

- *looks like linked list* since there is no limit on number of elements, but *not a linked list*
- implemented using arrays; create a new big one when original is full
- LIFO (Last-In First-Out) implementation.
- **The implementation gives you LIFO, called a stack!**
- **More on stacks in CO322.**

Use of BadLinkedList.java

```
class Main {
  public static void main(String [] args) {
    BadLinkedList list = new BadLinkedList();

    for(int i=0; i<30; i++) {
      Points p = new Points(i, i);
      list.add((Object)p); // bad casting
    }

    while(true) {
      Points p = (Points) list.remove(); // casting bad
      if(p == null) break;
      p.show();
    }
  }
}
```

Generics: Idea

Generic programming is a style of computer programming in which:

- algorithms are written in terms of types *to-be specified-later*, that are then
- *instantiated* when needed for specific types provided as *parameters*

Generics: Why

- Elimination of Casts.
- Stronger type checks at compile time:
  - Compile-time errors:
    - ★ Detected early
    - ★ Easier to fix
  - Run-time errors:
    - ★ Does not surface immediately
    - ★ Harder to find/trace
    - ★ Costs more!

Stack implementation

- FIFO data structure
- Data can be anything! (example: memory address of functions, points on a path ...)
- Casting is bad since it breaks type safety!
- Reuse the code without knowing the type

Other things to note:

- *Implementation vs. interface*
- *Use of public vs. private functions*

Stack implementation: Stack.java

```
public class Stack<T> { // give an instance to T when using
    Stack<String>
    private int curr;
    private int max;
    private T [] stack;

    private static final int block_size = 10;

    public Stack() {
curr = 0;
max  = block_size;
stack = (T[]) new Object[block_size];
    }
....
}
```

Stack implementation: Stack.java

```
// private functions, others need not know about this
private boolean isFull() { return curr == max; }

private void more() {
  int newSize = max + block_size;
  T[] newStack = (T[]) new Object[newSize];
  for(int i=0; i<this.max; i++)
  newStack[i] = this.stack[i];

  max = newSize;
  stack = newStack;
}
```

Stack implementation: Stack.java

```
// public, how the world sees me
public boolean isEmpty() { return curr == 0; }
public void push(T obj) {
  if(isFull()) more();
  stack[curr++] = obj;
}

public T pop() {
  if(isEmpty()) return null;
  return stack[--curr];
}
```

Using the stack
see UseStack.java

```
class UseStack {
  public static void main(String [] args) {
    // need a stack of points and strings
    Stack <Points> pntStack = new Stack <Points> ();
    Stack <String> strStack = new Stack <String> ();
    Points p;
    for(int i=0; i<20; i++) {// type safe
      pntStack.push(new Points(i,i));
      strStack.push("this is " + i);
    }
    while(true) {
      p = pntStack.pop();
      if(p == null) break;
      p.show();
      System.out.println(strStack.pop());
    } } }
```

Note: about code

```
class Stack<T> { /* stack of T can take many */

class Stack<T1, T2, ... // example of taking many

Stack <Points> pntStack = new Stack <Points> ();
// create a stack of points

Stack <String> strStack = new Stack <String> ();
//create a stack of strings

// once created type safety is enforced at compile-time

strStack.push(new Point(i,i));// will not work
```

Type Parameters

```
class Stack<T> { /* T is the type parameters *
```

- By convention, type parameter names are single, uppercase letters.
- Replace the type value with a concrete value (known as **parameterized** type
- Most commonly used type parameter names are:
  - E - Element (used extensively by the Java Collections Framework)
  - T - Type
  - N - Number
  - K - Key
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types

Tuple or pairs

Simple, yet useful idea: two values as one
Can be used to return more than two values from a function.

```
public class Pair <K,V>{ // Key (K) and Value (V)
    private K key;
    private V val;

    public Pair(K key, V val) {
      this.key = key;
      this.val = val;
    }

    public K getKey() { return this.key; }
    public V getVal() { return this.val; }
}
```

Nested generics
see StackOfPairs.java

```
public class StackOfPairs{
    public static void main(String [] args) {
Stack <Pair<String, Points>> stack = new Stack <Pair<String,
    Points>>();
String str = "This is point ";
Pair<String, Points> tuple;
for(int i=0; i<20; i++) {
    str += i;
    stack.push(new Pair(str, new Points(i,i)));
}

while(true) {
    tuple = stack.pop();
    if(tuple == null) break;
    tuple.getVal().show();
}  } }
```

Sorting method

type parameters in the definition of functions.

```
static void swap(int [] array, int i, int j) {
  int tmp = array[i];
  array[i] = array[j];
  array[j] = tmp;
}

public static void sort(int [] array) { // bubble sort
  for(int i=0; i < array.length; i++)
    for(int j=array.length - 1; j > i; j--)
      if(array[j] < array[j-1])
        swap(array, j, j-1);
}
```

Generic Methods
see GenFun.java

type parameters in the definition of functions.

```
static <T> void swap(T[] array, int i, int j) {
  T tmp = array[i];
  array[i] = array[j];
  array[j] = tmp;
}

// only works for types T where it extends Comparable class
public static <T extends Comparable<T>> void sort(T [] array) {
    // bubble sort
  for(int i=0; i < array.length; i++)
    for(int j=array.length - 1; j > i; j--)
      if(array[j].compareTo(array[j-1]) <= 0)
    swap(array, j, j-1);
}
```

Generics: Things to remember

- You cannot substitute a primitive type for the generic type parameter

```
int [] a = {1, 23, 3, 5, 1, 32, 4}; // will NOT work
sort(a);
```

- You cannot create an instance of a type parameter. For example, the following code causes a compile-time error

```
stack = (T[]) new Object[block_size]; // will work
stack = (T[]) new T[block_size]; // will NOT work
```

- Static fields of type parameters are not allowed! (why?)
- You cannot create arrays of parameterized types