CSC 347 - Concepts of Programming Languages

Parametric Polymorphism

Instructor: Stefan Mitsch

Learning Objectives

❓ Inheritance for container classes?

- Understand type parameters vs. inheritance

Monomorphic Linked Lists (C)

- Implement for each pointer type?

```c
typedef struct Node Node;
struct Node {
  int*  item;
  Node* next;
};

int* get_last (Node* xs) {
  while (xs->next != NULL) {
    xs = xs->next;
  }
  return xs->item;
}
```

- Can inheritance solve this?

```
abstract class Node {
  Node next;
  Node getLast() {
    Node xs = this;
    while (xs.next != null) xs = xs.next;
    return ???; // no xs.item!
  }
}
class IntNode extends Node {
  int item;
  // duplicate getLast in every concrete class?
}
```

❓ Benefits? Downsides?

Generic Linked Lists (C)

- Use type `(void*)` in C

```c
typedef struct Node Node;
struct Node {
  void* item;
  Node* next;
};

void* get_last (Node* xs) {
  while (xs->next != NULL) {
    xs = xs->next;
  }
  return xs->item;
}
```

❓ Benefits? Downsides?

Generic Linked Lists (C)

- No static protection against casts from `(void *)`

```c
typedef struct Node Node;
struct Node {
  void* item;
  Node* next;
};
int main () {
  int* p    = (int*)  malloc (sizeof(int));
  *p        = 2123456789;
  Node* xs  = (Node*) malloc (sizeof(Node));
  xs->next  = NULL;
  xs->item  = p;                // store int pointer
  double* q = get_last(xs);     // alias of p
  printf ("q=%f\n", *q);        // unsafe access
}
```

```
$ clang -m32 parametric-03.c && ./a.out
q=9662106905734617826804919238843065958 4.000000
```

Generic Linked Lists (Java)

- Generic list using subtype polymorphism

```
1  static class Node {
2    Object  item;
3    Node    next;
4  }
5
6  static Object getLast (Node xs) {
7    while (xs.next != null) {
8      xs = xs.next;
9    }
10   return xs.item;
11 }
```

Generic Linked Lists (Java)

- Subtype polymorphism
- `ClassCastException` better than unsafe access

```java
static class Node    {
  Object  item;
  Node    next;
}
public static void main (String[] args) {
  Integer p = Integer.valueOf(2123456789);
  Node xs   = new Node();
  xs.next   = null;
  xs.item   = p;                        // store Integer
  Double q  = (Double) getLast(xs);     // ClassCastException
  System.out.printf ("d=%f\n", q);      // unsafe access
}
```

```
$ javac Parametric2.java
$ java Parametric2
java.lang.ClassCastException: Integer cannot be cast to Double
        at Parametric.main(Parametric2.java:10)
```

Generic Linked Lists (Java)

- Generic list using parametric polymorphism

```
1  static Node<X> {
2    X        item;
3    Node<X> next;
4  }
5
6  static <X> X getLast (Node<X> xs) {
7    while (xs.next != null) {
8      xs = xs.next;
9    }
10   return xs.item;
11 }
```

Parametric Polymorphism

- Compiler errors better than runtime exceptions

```
1  static class Node<X> {
2    X       item;
3    Node<X> next;
4  }
5  public static void main (String[] args) {
6    Integer p = Integer.valueOf(2123456789);
7    Node<Integer> xs = new Node<>();
8    xs.next   = null;
9    xs.item   = p;                        // store Integer
10   Double q  = (Double) getLast(xs);     // compiler error
11   System.out.printf ("d=%f\n", q);      // unsafe access
12 }
```

```
1 $ javac Parametric1.java
2 error: incompatible types: Integer cannot be converted to Double
3   Double q = (Double) getLast(xs);   // compiler error
4                                ^
```

Java Type Parameter Erasure

- Java type parameters are not part of runtime type information

```
1  static class ArrayList<X> {
2    X[] a;
3
4    ArrayList(int n) {
5      a = new X[n];
6    }
7
8    void put (int i, X item) { a[i] = item; }
9    X    get (int i)         { return a[i]; }
10 }
```

```
1 $ javac Parametric3.java
2 error: generic array creation
3   a = new X[n];
4           ^
```

Java Type Parameter Erasure

- Cast is not checked at runtime

```
1  static class ArrayList<X> {
2    X[] a;
3
4    ArrayList(int n) {
5      a = (X[]) new Object[n];
6    }
7
8    void put (int i, X item) { a[i] = item; }
9    X    get (int i)         { return a[i]; }
10 }
```

```
1 $ javac -Xlint:unchecked Parametric4.java
2 warning: [unchecked] unchecked cast
3     a = (X[]) new Object[n];
4                 ^
```

❓ Why is cast not checked at runtime?

Java Type Parameter Erasure

- Okay to ignore, since array is empty

```
1  static class ArrayList<X> {
2    X[] a;
3
4    @SuppressWarnings("unchecked")
5    ArrayList(int n) {
6      a = (X[]) new Object[n];
7    }
8
9    void put (int i, X item) { a[i] = item; }
10   X    get (int i)         { return a[i]; }
11 }
```

```
1 $ javac -Xlint:unchecked Parametric4.java
```

Java Type Parameter Erasure

- Not possible to check casts when writing to unparameterized list

```
1 ArrayList<String> ss = new ArrayList<>(10);
2 ArrayList os = ss;
3 os.put (1, 2123456789);
4 String s = ss.get (1);
```

```
1 $ javac Parametric6.java
2 Note: Parametric6.java uses unchecked or unsafe operations.
3 Note: Recompile with -Xlint:unchecked for details.
```

```
1 $ java Parametric6
2 ClassCastException: Integer cannot be cast to class String
3   at Parametric.main(Parametric6.java:4)
```

Arrays Checked When Assigned

- Java stores types with arrays

```
1 String[] ss = new String[10];
2 Object[] os = ss;
3 os[1] = 2123456789;
4 String s = ss[1];
```

```
1 $ javac Parametric7.java
2 // no warnings
```

```
1 $ java Parametric7
2 ArrayStoreException: Integer
3    at Parametric.main(Parametric7.java:3)
```

Parametric Polymorphism

- In Java, C#, Ada, Haskell, Scala, Rust, etc
- First developed in ML (1970s, modern descendants: F#, oCaml, Standard ML)
- Strong connection to mathematical logic: Agda, Coq, etc.
- C++ templates are different
  - Template itself has no executable form: `List<T>`
  - Executable generated for each instantiation: `List<int>`, `List<string>`

Summary

- Parametric polymorphism allows independence between container class and types of contained objects
- Type safety when reading from and writing to containers
- Not all languages retain type parameters at runtime (Java: compiler knows generics, JVM does not)