CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Introduction

CS Workshop: Types & Generics

Objectives

- Explain the precise notions of types, type conflicts and type safety.
- Distinguish between static and dynamic type safety.
- Distinguish between types, classes and interfaces in Java.
- Recognize the special role of containers for type safety
- Explain the notions of a generic class/interface and type parameters
- Construct simple generic classes to use in a sample project
- Describe the distinction between Java Generics and C++ Templates

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Assumed

- Java Classes & Interfaces
- Polymorphism, C++ virtual

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

A. Types

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Types?

A **data type** or simply **type** is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Using data?

- Numbers: Do arithmetic, compare by size, …
- Strings: Concatenate, split, search, determine length, …
- Employees: Get or set name, department, salary, …

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Using data?

Compiler seems to detect type errors! →

```
class Employee {
    private String name;
    private String department;

    public Employee(String name, String department) {
    public String getName() {
    public void setName(String name) {
    public String getDepartment() {
    public void setDepartment(String department) {
}

public class TypeError {
    public static void main(String[] args) {
        int x = 2,y = 3;
        Employee e1 = new Employee("Hugo","Sales"),
                 e2 = new Employee("Helga","Management");

        System.out.println(x + y);
        System.out.println(e1 + e2);

        System.out.println(e1.getName());
        System.out.println(x.getName());
    }
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Type errors:

Programming errors that perform or imply improper use of data because of erroneous assumptions about its type

- Arithmetic on non-numeric data
- Illegal method calls
- …

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Compared to Python:

```
1   import employee
2
3   def main():
4       e1 = employee.Employee('Hugo', 'sales')
5       e2 = employee.Employee('Helga', 'management')
6
7       print(e1.get_name())
8
9       print(e1 + e2)
10
11
12  main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
bernd@ubuntu:~/work$ python main.py
Hugo
Traceback (most recent call last):
  File "main.py", line 12, in <module>
    main()
  File "main.py", line 9, in main
    print(e1 + e2)
TypeError: unsupported operand type(s) for +: 'instance' and 'instance'
```

Type errors detected at run time →

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Type safety:

Type safety is the extent to which a programming language discourages or prevents type errors.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

When to detect typing errors:

- Java example: at compile-time

  static: based on the source code
- Python example: at run-time

  dynamic: based on program behavior

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Compare to

- Compile-time errors

  based on analysis of the source code
- Run-time errors

  based on program behavior

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Check Point 1:

```
a)  String name = "Hugo";
b)  int l1 = name.length();
c)  String fullName = name + " Halfling ";
d)  l1 = l1 + " Halfling ".length();
e)  fullName = fullname + l1;
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Research problem 1:

Can the Java compiler detect every possible type conflict?

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Types, Classes and Interfaces:

Classes and Interfaces define **how** an instance behaves (what and with which parameters a method can be called and what is returned)

Classes also define **how it actually does it**.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

```
public class Cat {
    private String sound;
    public Cat(String sound) {
        this.sound = sound;
    }
    public String getSound() {
        return sound;
    }
}

public interface Sounding {
    String getSound();
}
```

Both:

"Whatever the instance is, you can call **getSound()** on it and get a String back!"

The class describes **how!**

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Types, Classes and Interfaces:

→ Classes and Interfaces are types as well!

```
Employee hugo = new Employee(...);
```

Type

Constructor call

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Subtypes:

- Class Cat **extends** class Pet
- Interface SoccerMatch **extends** Interface Event
- Class Bicyle **implements** Interface Vehicle

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Substitution

Example: **Cat** is subtype of **Pet**

Pet p = new **Cat**(...) ← works

Barbara Liskov

Liskov Substitution Principle:

You can substitute an object of a subtype for an object of the supertype without breaking the program.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Check Point 2:

// Assume Cat is a subtype of Pet
// Assume MaineCoon is a subtype of Cat

```
a)  Cat c = new Cat(...);
b)  Cat c = new MainCoon(...);
c)  Pet p = new MainCoon(...);
d)  MainCoon m = new Cat(...);
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

B. Container classes

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Container Classes: Basics

A container is a class, a data structure, [...] or an abstract data type (ADT) whose instances are collections of other objects.

Characterized by access, storage and traversal

UNIVERSITY of WEST FLORIDA

Container Classes: Characterization

- access, [...] the way of accessing the objects of the container. In the case of arrays, access is done with the array index. In the case of stacks, access is done according to the LIFO (last in, first out) order [...]
- storage, [...] the way of storing the objects of the container;
- traversal, [...] the way of traversing the objects of the container.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Container Classes: Element types

- The elements of a **homogeneous** container are all of the same type (all Strings, all Employees, …)

- The elements of a **heterogeneous** container can be of different types (Strings, Employees, etc - all mixed)

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Container Classes: Heterogeneous containers

Programmer must know/determine what element type an element has, before operating with it

Compiler has no idea ← **No static type safety**

Doing that wrong/forgetting it can result in a type violation!

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Container Classes: Homogeneous containers

- Programmer knows what type an element has
- Compiler knows what type an element has

  → **Static type safety**

We have to tell the compiler about the element types

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Container Classes: ArrayLists

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified [...] Elements can be added and removed from an ArrayList whenever you want

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

ArrayLists: heterogeneous (Raw Type)

ArrayLists store any kind of object!

```
ArrayList pets = new ArrayList();
pets.add(new Pet());
pets.add(new Scanner(System.in));
pets.add("car");
```

This all works!

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

ArrayLists: homogenous (-ish)

Type parameter

ArrayList<String> can **only** store instances of String

ArrayList<Employee> can **only** store instances of Employee

ArrayList<Pet> can **only** store instances of Pet

Diamond

```
ArrayList<String> list = new ArrayList<>();
```

Type

Constructor call

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

ArrayLists: homogenous


```
ArrayList<Pet> pets = new ArrayList<>();
pets.add(new Pet());                          This works!
pets.add(new Scanner(System.in));             This doesn't!
pets.add("car");                              This doesn't either!
```

Rule: You can store anything that can be assigned to a variable of type Pet

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

ArrayLists: homogeneous (?)

Work with the Substitution Principle

```
ArrayList<Pet> pets = new ArrayList<>();
pets.add(new Pet());
pets.add(new Cat());
pets.add(new Dog());
```

This all works!

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

ArrayLists: homogeneous (not really)

But what I **want** to store Pets, Scanners and Strings???

```
ArrayList<Object> pets = new ArrayList<>();
pets.add(new Pet());
pets.add(new Scanner(System.in));
pets.add("car");
```

This all works!

But is it a good idea?

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Check Point 3:

// Assume Cat is a subtype of Pet
// Assume MaineCoon is a subtype of Cat

```
ArrayList<Cat> catShelter = new ArrayList<>();
a)  catShelter.add(new MainCoon(...))
b)  catShelter.add(new Pet(...))
c)  catShelter.add(new Cat(...))
d)  catShelter.add("Here, Kitty, Kitty")
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

C. Generics

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics

A generic type is a generic class or interface that is parameterized over types.

The Java™ Tutorials

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Generics

are somewhere in between

- homogeneous (restricted to one type, e.g. **String**)
- heterogeneous (**Object** type parameter)

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

What? Generic Interfaces?

```
public interface Iterable<T>
public interface Collection<E> extends Iterable<E>
public interface List<E> extends Collection<E>
public class ArrayList<E>
   extends AbstractList<E>
   implements List<E>, RandomAccess,
              Cloneable, Serializable
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Generic Interfaces

Like always: Use Interfaces to detach the core code from implementation details

```
// Not recommended if you don't need ArrayList specific features
ArrayList<String> ls = new ArrayList<>();

// In many (most?) cases: Better
List<String> ls = new ArrayList<>();
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics and subtyping

Just the facts - no deep explanation, why!

- ArrayList<Pet> is subtype of AbstractList<Pet>
- ArrayList<Pet> is subtype of List<Pet>
- ArrayList<Cat> is NOT subtype of ArrayList<Pet> or List<Pet> ...

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

D. Generics Project

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project

Shows the two main “use cases” for Generics

- Wrappers (Classes “around” objects)
- Collections

CS Workshop: Types & Generics

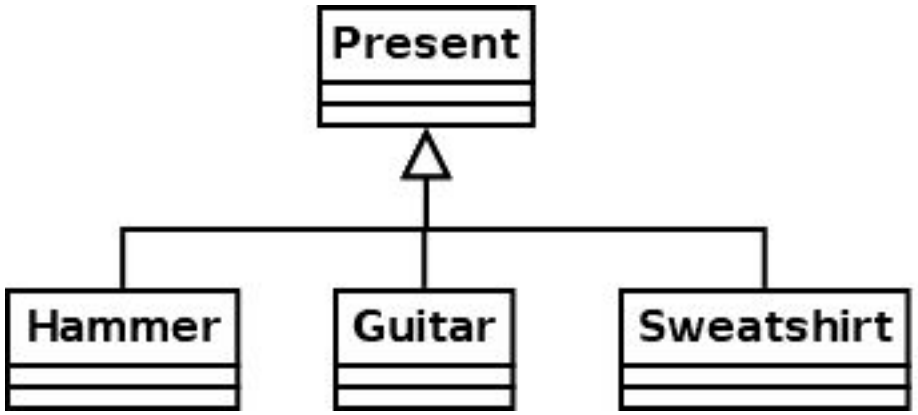
UNIVERSITY of WEST FLORIDA

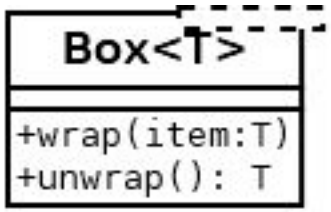
Java Generics Project : Wrapping presents

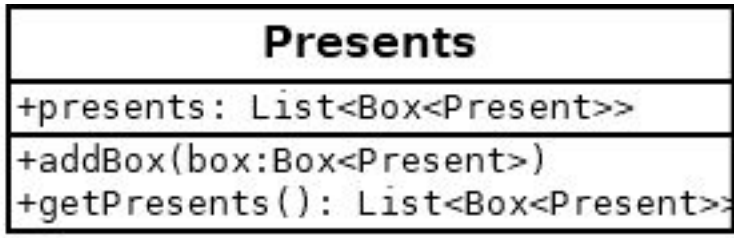
CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics (Case study)







CS Workshop: Types & Generics

Java Generics Project

Step 1: Objects and Wrappers

CS Workshop: Types & Generics

Java Generics Project: Presents

```
public abstract class Present {}

public class Guitar extends Present {
   @Override
   public String toString() {
      return "A guitar - Hopefully a 1956 Les Paul!";
   }
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Boxes (Wrappers)

```
public static void main(String[] args) {
    Present p1 = new Guitar();
    Box<Present> box = new Box<>();
    box.wrap(p1);
    Present p2 = box.unwrap();
    System.out.println(p2);
}
```

→ A guitar - Hopefully a 1956 Les Paul!

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project

Step 2: Collections of Generics

(Little tricky!)

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Boxes (Wrappers)

```
public class Box<T> {
   private T item;
   public Box() {}
   public Box(T item) { wrap(item); }

   public void wrap(T item) { this.item = item; }
   public T unwrap() { return item; }
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Presents (Container)

```
public class Presents {
   private List<Box<Present>> presents;
   public Presents() {
      presents = new ArrayList<>(); }
   public void addBox(Box<Present> b) {
      presents.add(b); }
   ...
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Presents (Container)

```
public class Presents {
   ...
   public List<Box<Present>> getPresents() { // Copy!
      List<Box<Present>> result = new ArrayList<>();
      for (Box<Present> box : presents) {
                        result.add(box); }
      return result;
   }
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Main

```
public static void main(String[] args) {
   Presents list = new Presents();
   System.out.println("Wrapping a guitar");
   list.addBox(new Box<Present>(new Guitar()));
   System.out.println("Wrapping a hammer");
   list.addBox(new Box<Present>(new Hammer()));
   System.out.println("Wrapping a sweatshirt");
   list.addBox(new Box<Present>(new Sweatshirt()));
   ...
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics Project: Main

```
public static void main(String[] args) {
    ...
    // Now see what's stored
    List<Box<Present>> boxes = list.getPresents();
    for (Box<Present> box : boxes) {
         System.out.println(box.unwrap());
    }
}
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

E. Generics & Templates

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics & C++ Templates

Main difference is semantics:

Type Erasure vs. compile-time translation

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics & C++ Templates

Java type erasure: The process of enforcing type constraints only at compile time and discarding the element type information at runtime. Uses raw versions at runtime!

Artificial access methods (bridges) are using static type checking for parameters and upcasting for return values - hiding the raw storage.

CS Workshop: Types & Generics

Java Generics & C++ Templates

C++ template translation: The compiler doesn't compile a template. Once it's instantiated with a type parameter, the compiler generates source code by literally substituting the actual type for the type parameter, then compiling it, then creating the instance from the compiled class.

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Java Generics & C++ Templates

C++ template translation:

```
template<typename T>
class Foo
{
public:
  T& bar()
  {
      return subject;
  }
private:
  T subject;
};
```

+

```
Foo<int> fooInt;
```

=

```
class FooInt
{
public:
  int& bar()
  {
      return subject;
  }
private:
  int subject;
}
```

+

```
Fooint fooInt;
```

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

C++ Templates

Lots of code → Demo

Template1.cpp:	Simple template with different instances
Template2.cpp:	Same template with polymorphic class instances (using pointers, of course)

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

C++ Templates complications (Template3)

Pointer types: Array<Pet>

Array<Pet*>

Array<Pet>*

Array<Pet*>*

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

C++ Templates complications

Pointer types & initialization:

```
Array<Pet> ap;
Array<Pet*> app;
Array<Pet>* app1 = new Array<Pet>;
Array<Pet*>* app2 = new Array<Pet*>;
```

F. Conclusion

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Generics & Templates

Same “use cases” → Wrappers, containers, ...

Same idea → Static type safety for containers

Similar syntax → Name<Type>

Similar properties → Between homogenous and heterogeneous (polymorphism)

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Generics & Templates

Different semantics → Type erasure, compiler translation

Different problems → Subtyping, pointers

CS Workshop: Types & Generics

UNIVERSITY of WEST FLORIDA

Next studies

Subtyping Generics: Still open!

→ Wildcards, upper/lower bounds

Complex use cases: Parallel class hierarchies,

Merged class hierarchies, ...