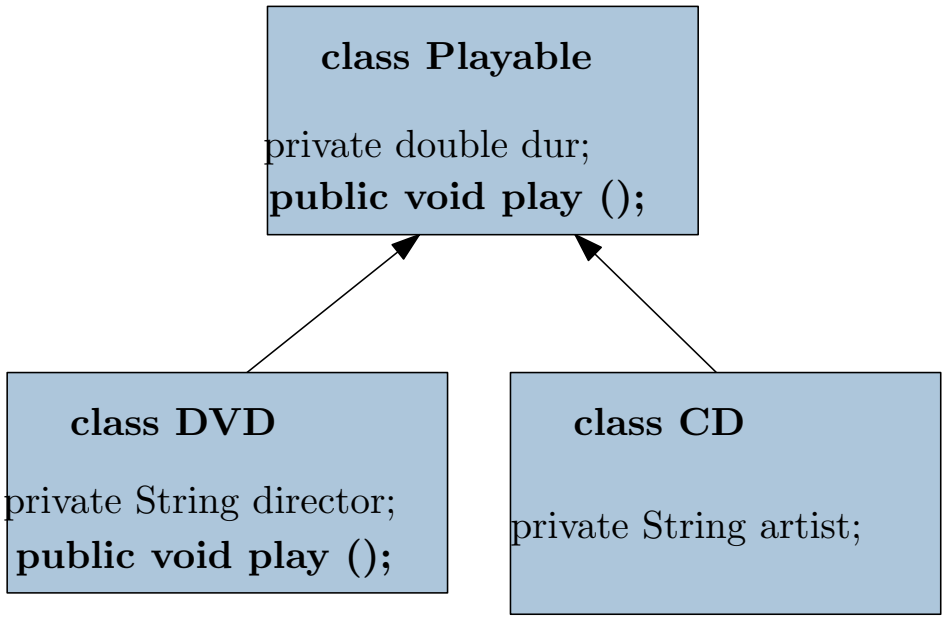
Generics

Daniel Archambault

Previously in CS-115



Inheritance II

Previously in CS-115

- Why is inheritance important?

Previously in CS-115

- Why is inheritance important?
- What is overloading? What is overriding?

Previously in CS-115

- Why is inheritance important?
- What is overloading? What is overriding?
- What purpose does casting serve? How is it related to `Object`?

Previously in CS-115

- Why is inheritance important?
- What is overloading? What is overriding?
- What purpose does casting serve? How is it related to `Object`?
- How can the `super` keyword be used (two ways)?

Previously in CS-115

- Why is inheritance important?
- What is overloading? What is overriding?
- What purpose does casting serve? How is it related to `Object`?
- How can the `super` keyword be used (two ways)?
- What is an abstract class? When do we use abstract methods?

Previously in CS-115

- Why is inheritance important?
- What is overloading? What is overriding?
- What purpose does casting serve? How is it related to `Object`?
- How can the `super` keyword be used (two ways)?
- What is an abstract class? When do we use abstract methods?
- What is an interface?

Previously in CS-115

- Inheritance occasionally is too flexible
- Stronger typing please

Generics

The Problem with `Object`

- Inheritance is great and allows for some type constraints
- But, what if we want to...
  - create generic code
  - **and** ensure the existence of operations
- Currently, we only have `Object` at our disposal
- This is where the problem lies

List of Products

- Suppose we want to organise our products into lists

```
public class ListOfProducts {
    private Products[] products;

    public Product getElementAt (int i) {
        ...
    }

    public void addProduct (int i, Product p) {
        ...
    }
}
```

- This works great for `Product` (and its children)
- It does not work for anything outside `Product`
- Operations don't *really* have anything to do with `Product`

List of Strings

- This is really useful code!
- Okay, I want to reuse this class for strings now

```
public class ListOfStrings {
  private String[] strs;

  public String getElementAt (int i) {
      ...
  }

  public void addString (int i, String p) {
      ...
  }
}
```

- Right, now I need to reimplement everything...
- This is not good. `Idea` - reimplement, `Lemon` - reimplement ...

Object for the Win!

- To avoid reimplementation, let's try and use Object

```
public class ListOfObjects {
  private Object[] objs;

  public Object getElementAt (int i) {
      ...
  }

  public void addObject (int i, Object p) {
      ...
  }
}
```

- This is perfect! Now, we can put any class we want in!
- What could possibly be the problem?

The Outside World is Cruel

- Let's start treating ListOfObjects as an ADT

```
...
ListOfObjects objs = new ListOfObjects (); //empty
objs.addObject (0, new String ("Hi there"));
objs.addObject (1, new Boolean (false));
objs.addObject (1, new DVD ("Love Actually"));
objs.addObject (2, new ListOfObjects ());
objs.addObject (4, new Lemon ("Neal Harman"));
objs.addObject (5, new KitchenSink ());
...
Object o4 = objs.getElementAt (4); //What the &^#%?
...
```

- We have no idea what the type of `o4` is
- We could figure it out (use `instanceof`), but *lots* of cases to handle!

Generics Provide a Better Way

- Specify a type parameter and then it behaves that way

```
public class ListOfThings<T> {
   private T[] things;

   public T getElementAt (int i) {
        ...
   }

   public void addThing (int i, T p) {
        ...
   }
}
```

- This puts restrictions on the type that can be put into the list

Type is now constrained

- By specifying the type parameter in the outside world, we constrain how the list can be used.

```
ListOfThings<String> strs =
        new ListOfThings<String> ();
ListOfThings<Product> prdcts =
        new ListOfThings<Product> ();
strs.addThing (0, new String ("hi"));
prdcts.addThing (0, new DVD ("Love Actually");
//does not compile
prdcts.addThing (0, new String ("Works?"));
...
strs.getElementAt (4); //always string
prdcts.getElementAt (4); //always product
```

- We have type consistency without reimplementation.

Generic for a Method Only

- Generics can be defined locally for one method
- Very useful for static methods

```
public class MyClass {
   ...
   //The type parameter T defined for tryThis only
   public static <T> void tryThis () {

   }
}
...

ListOfObjects.<String>tryThis (); //Calls on String
```

Limitations: No `new` for type variable

- Within a generic cannot instantiate the type variable

```
public class ListOfThings<T> {
    ...
    T t = new T(); //compile error
    T[] ts = new T[4]; //compile error
}
```

- For simplicity, construct objects in the outside world
- Pass the references to the inside world.

Limitations: No Simple Types

- In the outside world, type parameters cannot be simple types

```
//compile error
ListOfThings<int> noWork = new ListOfThings<int> ();

//compile error
ArrayList<int> arr = new ArrayList<int> ();
```

- Simple types cannot be a type parameter
- Only class types can be type parameter (capital letters)

Hey, I've already seen this!

- You have already seen generics with `ArrayList`
  - ArrayList<String> a = new ArrayList<String> ();
  - an extensible list of strings and no other types
- Java uses generics often to group things together
- They can be very useful for data structures

How does this work?

- `ArrayList` grows as it gets full
- How can we do this without `T[] arr = new T[4]`?
- You can take open Java apart!
  - Go look inside openjdk and open `ArrayList.java`

How does this work?

- `ArrayList` grows as it gets full
- How can we do this without `T[] arr = new T[4]`?
- You can take open Java apart!
  - Go look inside openjdk and open `ArrayList.java`
- `Object[] elementData;`
  - Hey! It's using Object. No fair!
  - BUT: Template parameter enforces type.

Quiz

Quiz

- What is `Object` and its main advantage/disadvantage

Quiz

- What is `Object` and its main advantage/disadvantage
- Why do generics help with this problem?

Quiz

- What is `Object` and its main advantage/disadvantage
- Why do generics help with this problem?
- How do you specify generic types?

Quiz

- What is `Object` and its main advantage/disadvantage
- Why do generics help with this problem?
- How do you specify generic types?
- Can you do it for a single method?

Quiz

- What is `Object` and its main advantage/disadvantage
- Why do generics help with this problem?
- How do you specify generic types?
- Can you do it for a single method?
- Can you instantiate them (`new T[4]` or `new T()`)