Lecture 11: Using Generic Classes

AITI Nigeria Summer 2012
University of Lagos.

http://aiti.mit.edu
Library Classes and Casting

- ArrayList is an implementation of a growable array in Java
  - java.util.ArrayList

- Java 1.3 implementation of ArrayList stored Objects
  - Remember that all classes inherit from the Object class
  - Can store any class!
ArrayList example

ArrayList contactList = new ArrayList();

Person p = new Person("Mike", "123456", 30);
Business b = new Business("Debonaires", "23234");

contactList.add(p);
contactList.add(b);

((Person)contactList.get(0)).print();
Casting

```java
((Person) contactList.get(0)).print();
```

- `contactList.get(0)` returns a reference of type Object to a Person (“Mike”);
- `(Person)` casts that Object reference to a reference of type Person.
Casting

((Person)contactList.get(0)).print();

- contactList.get(0) returns a reference of type Object to a Person (“Mike”);
- (Person) casts that Object reference to a reference of type Person.
- ((Person)contactList.get(0)) evaluates to a reference to a Person.
Casting

((Person)contactList.get(0)).print();

• contactList.get(0) returns a reference of type Object to a Person (“Mike”);
• (Person) casts that Object reference to a reference of type Person.
• ((Person)contactList.get(0)) evaluates to a reference to a Person.
• .print() calls print on the Person.
Without Casting?

```java
contactList.get(0).print();
```

- This would fail at compile time because:
  - `ContactList.get(0)` returns an `Object` reference
  - An `Object` does not define the `print()` method
Using Inheritance

\((\text{Contact})\text{contactList.get(0)})\).print();

- Contact defines print(), so we can use Inheritance and cast to the superclass.
  - Person and Business override print()
- Java will call print in Person using dynamic dispatch.
Using Inheritance

• We can write code to print out all Contacts:

```java
for (int j = 0; j < contactList.size(); j++) {
    ((Contact)contactList.get(j)).print();
}
```

• But we still have to cast

• And we still have to remember that contactList stores Contacts.
No Error Compiler Error Catching

What if we wrote:

```java
contactList.add(new Person(…));
contactList.add(new String(“”));
```
String extends Object so this is legal.

Now:

```java
for (int j = 0; j < contactList.size(); j++) {
    ((Contact)arrayList.get(j)).print();
}
```
On 2nd iteration, the cast will fail!
A Contact reference cannot be String!
Generics in Java

• The solution to our problems!
• We can define that ArrayList stores only Contacts and subclasses of Contact:

```java
ArrayList<Contact> = new ArrayList<Contact>();
```

• This is read “ArrayList of Contacts”
• Now we do not have to cast!
• And we have error checking!
ArrayList can be provided with an optional type. The type of Object it stores.

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all element interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This unsynchronized.)

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation run requires $O(n)$ time. All of the other operations run in linear time (roughly speaking). The constant factor is low con
Generics in Java

GenericClass<Type> gc = new GenericClass<Type>();

- The type specified can either be an interface or a class.

- Reference declaration (left side) has to match object created (right side)
ArrayList ContactList
with Generics

ArrayList<Contact> contactList =
    new ArrayList<Contact>();

Person p = new Person("Mike", "123456", 30);
Business b = new Business("Debonaires",
    "2323323234");

contactList.add(p);
contactList.add(b);

contactList.get(0).print();
Using Generics

• We can write code to print out all Contacts:

```java
for (int j = 0; j < contactList.size(); j++) {
    contactList.get(j).print();
}
```

• No casting!
• Less typing!
Compiler Error Catching

```java
ArrayList<Contact> = new ArrayList<Contact>();

contactList.add(new Person(...));
contactList.add(new String(""));
```

This statement will not compile! Because String is not a Contact!
Java Generics

• Use Generics everywhere you can!

• This was a gentle introduction to Generics
  – You can define your own Generic classes but we did not cover this.
  – Generics can get very complex.