**Topic 6**
**Generic Data Structures**

"Get your data structures correct first, and the rest of the program will write itself."
- *David Jones*

---

**Back to our Array Based List**
- Started with a list of ints
- Don't want to have to write a new list class for every data type we want to store in lists
- Moved to an array of `Objects` to store the elements of the list
  ```java
  // from array based list
  private Object[] myCon;
  ```

---

**Using Object**
- In Java, all classes inherit from exactly one other class except `Object` which is at the top of the class hierarchy
- Object variables can refer to objects of their declared type and any descendants
  - polymorphism
- Thus, if the internal storage container is of type `Object` it can hold anything
  - primitives handled by *wrapping* them in objects.
    - `int` - `Integer`, `char` - `Character`

---

**Difficulties with Object**
- *Creating* generic containers using the Object data type and polymorphism is relatively straightforward
- Using these generic containers leads to some difficulties
  - Casting
  - Type checking
- Code examples on the following slides
Clicker Question 1

- What is output by the following code?
  ```java
  GenericList list = new GenericList(); // 1
  String name = "Olivia";
  list.add(name); // 2
  System.out.println( list.get(0).charAt(2) ); // 3
  
  A. i
  B. No output due to syntax error at line // 1
  C. No output due to syntax error at line // 2
  D. No output due to syntax error at line // 3
  E. No output due to runtime error.
  ```

Code Example - Casting

- Assume a list class
  ```java
  GenericList li = new GenericList();
  li.add("Hi");
  System.out.println( li.get(0).charAt(0) );
  // previous line has syntax error
  // return type of get is Object
  // Object does not have a charAt method
  // compiler relies on declared type
  System.out.println(
    ((String)li.get(0)).charAt(0) );
  // must cast to a String
  ```

Code Example – type checking

```java
// pre: all elements of li are Strings
public void printFirstChar(GenericList li) {
    String temp;
    for(int i = 0; i < li.size(); i++) {
        temp = (String)li.get(i);
        if( temp.length() > 0 )
            System.out.println(
                temp.charAt(0) );
    }
  }
// what happens if pre condition not met?
```
Is this a bug or a feature?

"Fixing" the Method

```java
// pre: all elements of li are Strings
public void printFirstChar(GenericList li) {
    String temp;
    for(int i = 0; i < li.size(); i++) {
        if( li.get(i) instanceof String ) {
            temp = (String)li.get(i);
            if( temp.length() > 0 )
                System.out.println(temp.charAt(0));
        }
    }
}
```

Generic Types

- Java has syntax for parameterized data types
- Referred to as Generic Types in most of the literature
- A traditional parameter has a data type and can store various values just like a variable
  ```java
  public void foo(int x)
  ```
- Generic Types are like parameters, but the data type for the parameter is data type
  - like a variable that stores a data type
  - this is an abstraction. Actually, all data type info is erased at compile time

Making our Array List Generic

- Data type variables declared in class header
  ```java
  public class GenericList<E> {
  ```
- The `<E>` is the declaration of a data type parameter for the class
  - any legal identifier: Foo, AnyType, Element, DataTypeThisListStores
  - Sun style guide recommends terse identifiers
- The value E stores will be filled in whenever a programmer creates a new `GenericList`
  ```java
  GenericList<String> li =
      new GenericList<String>();
  ```
Modifications to GenericList

- instance variable
  private E[] myCon;
- Parameters on
  – add, insert, remove, insertAll
- Return type on
  – get
- Changes to creation of internal storage container
  myCon = (E[])new Object[DEFAULT_SIZE];
- Constructor header does not change

Using Generic Types

- Back to Java's ArrayList
  ArrayList list1 = new ArrayList();
  – still allowed, a "raw" ArrayList
  – works just like our first pass at GenericList
  – casting, lack of type safety

Using Generic Types

ArrayList<String> list2 =
  new ArrayList<String>();
  – for list2 E stores String
list2.add( "Isabelle" );
System.out.println(
  list2.get(0).charAt(2) ); //ok
list2.add( new Rectangle() );
  // syntax error
Parameters and Generic Types

- Old version
  //pre: all elements of li are Strings
  public void printFirstChar(ArrayList li){

- New version
  //pre: none
  public void printFirstChar(ArrayList<String> li){

- Elsewhere
  ArrayList<String> list3 = new ArrayList<String>();
  printFirstChar( list3 ); // ok
  ArrayList<Integer> list4 = new ArrayList<Integer>();
  printFirstChar( list4 ); // syntax error

Generic Types and Subclasses

ArrayList<Shape> list5 =
  new ArrayList<Shape>();
list5.add( new Rectangle() );
list5.add( new Square() );
list5.add( new Circle() );
// all okay

- list5 can store Shape objects and any
descendants of Shape