
CS206

More Inheritance
Generics

try/catch — with resources

```
public void readOneLineTC(String filename)
{
    BufferedReader br;
    try {
        br = new BufferedReader(
            new FileReader(filename));
        br.readLine();
    } catch (FileNotFoundException fnf) {
        System.err.println("No file " + e);
    } catch (IOException e) {
        System.err.println("Reading " + e);
    } finally {
        if (br!=null) {
            try {
                br.close();
            } catch (IOException ioe) {
                System.err.println("Close" + ioe);
            }
        }
    }
}
```

```
public void readOneLineTCR(String filename)
{
    try (BufferedReader br = new BufferedReader(
        new FileReader(filename));) {
        br.readLine();
        // close unnecessary in this formulation
    } catch (FileNotFoundException e) {
        System.err.println("Open " + e);
    } catch (IOException e) {
        System.err.println("Reading " + e);
    }
}
```

finally == code that WILL be executed

Close can throw an exception so it too must be caught

See FileOpen.java

Method Overriding

- Inherited methods from the superclass can be redefined/changed
 - “signature” stays the same
 - signature = name+type of all args
- The appropriate version to call is determined at run time
- Most common overrides
 - `toString`
 - `equals`

DogDriver

DogDriver.java

Parsing strings

- **Split method of String**

```
string.split(String regexp)
```

- split a string into an array of Strings based on matching delimiter. Then go through the array appropriately

```
StringSplitter.java
```

Generics

- A way to write classes and methods that can operate on a variety of data types without being locked into specific types at the time of definition
- Write definitions & implementations with “Generic” parameters
- The generics are instantiated (locked down) when objects are created

Generic Methods

```
import java.util.Random;
/*****
 * @author gTowell
 * Created: August 28, 2019
 * Modified: Jan 24, 2019
 * Purpose:
 * Generic Methods
 *****/
public class GenericMethod {
    public static void main(String[] args) {
        Integer[] jj = { 1, 2, 3, 4, 5, 6, 7, 8, 9 }; // NOTE AUTOBOXING!!!
        new GenericMethod().randomize(jj);
        for (int j : jj)
            System.out.println(j);
        String[] ss = { "A", "B", "c", "d", "E", "F" };
        new GenericMethod().randomize(ss);
        for (String s : ss)
            System.out.println(s);
    }

    public <T> void randomize(T[] data) {
        Random r = new Random();
        for (int i = 0; i < data.length; i++) {
            int tgt = r.nextInt(data.length);
            swap(data, tgt, i);
        }
    }
}
```

— generic swap method
— use reflection to check class

Generic Class

```
import java.io.BufferedReader;
import java.io.StringReader;
/**
 * Simple generic class example
 * @author gtowell
 *
 * @param <A>
 */
public class GenericClass<A> {
    /** A non-generic value */
    private double amount;
    /** A generic value */
    private A otherValue;
    /**
     * Constructor.
     * @param other the generic value
     * @param amt a double value.
     */
    public GenericClass(A other, double amt) {
        this.otherValue = other;
        this.amount = amt;
    }
    public static void main(String[] args) {
        GenericClass<String> gString = new GenericClass<String>("ASDF", 24.5);
        System.out.println(gString);
        GenericClass<Double> gDouble = new GenericClass<Double>(99.5, 44.5);
        System.out.println(gDouble);
        GenericClass<BufferedReader> gBR = new GenericClass<BufferedReader>(
            new BufferedReader(new StringReader("When in the course")), 99.8);
        System.out.println(gBR);
    }
}
```

write a toString function
for this class

Generics Restrictions

- No instantiation with primitive types
 - `Genre<Double>` ok, but
`Genre<double>` is not
- Can not declare static instance variables of a parameterized type
- Can not create arrays of parameterized types
 - but you can create an array of `Object` then cast
 - `(T[]) new Object[10]`

Nested Class

- A class defined inside the definition of another class
- When defining a class that is strongly affiliated with another
 - help increase encapsulation and reduce undesired name conflicts.
- Nested classes are a valuable technique when implementing data structures
 - represent a small portion of a larger data structure
 - an auxiliary class that helps navigate a primary data structure
 - **ONLY** place that public instance variables are acceptable
 - They aren't really public

Nested Class Example

ClassNester.java