CMPE212 – Reminders

• Assn 4 is posted. Due a week from this Friday.

Today

• A Generic Factory Demo.
• The Class<T> object.
• Lambda Functions.
• Method References.

Generic Factory Demo

• Generate and return an ArrayList<T> given a Class object for the type and a text file of raw arguments.
• Make the entire factory program generic so it will work for any type.
• Assumptions:
  – The type has a constructor that matches the order of the arguments supplied in the file.
  – Arguments are of a numeric type or a String type.
• With some effort you could add code to get around these assumptions.
• See GenericFactory.java along with TestFactory.java.

The Class<T> Object

• You can pass a type into a method using a Class<T> object.
• The Object class contains a method .getClass() that returns a Class<T> object. Or you can just use the .class literal on a type.

```java
String aString = "Hello!";
System.out.println(aString.getClass());
System.out.println(String.class);
```

• Shows:
  ```java
class java.lang.String
class java.lang.String
```

The Class<T> Object, Cont.

• Everything in Java has its own Class object, even primitive types.

• See TestClass.java

• See the API for lots of methods belonging to this Object. Useful for determining all kinds of information about a type or an instance.

• The starting point for the use of Reflection.

From Before: Anonymous Class Example

```java
public class AnonymousClassDemo {
    public static void main(String[] args) {
        MessageSender ms = new MessageSender() {
            public void sendGreeting (String name) {
                System.out.println("Hello " + name + ":");
            } // end sendGreeting
        }; // end ms
        ms.sendGreeting("Steve");
    // end main
    } // end AnonymousClassDemo
```
Anonymous Class Example - Cont.
• MessageSender is an interface, not an Object:
  public interface MessageSender {
      void sendGreeting (String name);
  } // end MessageSender interface

• Java ≥ 8 now has a very tidy solution to using messy anonymous classes – Lambda Functions:
  public class LambdaDemo {
      public static void main(String[] args) {
          MessageSender ms = name -> System.out.println("Hello " + name);
          ms.sendGreeting("Steve");
      } // end main
  } // end LambdaDemo

The Lambda Version
public class LambdaDemo {
  public static void main(String[] args) {
      MessageSender ms = name -> System.out.println("Hello " + name);
      ms.sendGreeting("Steve");
  } // end main
} // end LambdaDemo

A Lambda Function
• Kind of like an "anonymous method".
• Syntax:
  – Parameters for the method first. No parameters, use { }, one parameter by itself – no brackets required, more than one use (a, b, etc.). No types required.
  – Then the fancy -> "arrow".
  – Then the method body. Put more than one statement inside { }.

• Could even define an inner interface:

The Lambda Version, Cont.
public class LambdaDemoAgain {
  interface MessageSender {
      void sendGreeting(String aName);
  } // end MessageSender interface

  public class LambdaDemoAgain {
      public static void main(String[] args) {
          MessageSender ms = name -> System.out.println("Hello " + name);
          ms.sendGreeting("Steve");
      } // end main
  } // end LambdaDemoAgain

Lambda Functions, Cont.
• Note how the abstract method in the interface determines the structure of the lambda function – the parameter and parameter type, the method name and the lack of a return statement.
• These functions could be useful! (Especially when attaching events to GUI components.)
• Only certain interface structures can be used.

Lambda Functions, An Example
• Suppose you want to have a method that only displays certain members of a collection, depending on a criteria that is specified outside the method and provided as an argument.
  • You don’t want to hard code the criteria inside the display method.
  • First technique: Supply an object implementing an interface that has a "filter" method that returns a true or false. See TestFilter1.java
Lambda Functions, An Example, Cont.

- Second technique: TestFilter2.java – use an anonymous class instead.
- This is the best version, so far!

Lambda Functions, Cont.

- But how does it work?
- The structure of the method implemented by the lambda function is specified by the interface type used in the displaySome method. The signature is:
  ```java
  boolean check(Person)
  ```
- The compiler knows from this that the type to the left of the `->` must be a `Person` and the expression to the right of the `->` must evaluate to a `boolean`.

Lambda Functions, Cont.

- Filter is an example of a `Functional Interface`.
- These interfaces can only contain a single abstract method.
- Lambdas can only be created using `Functional Interfaces`.
- You can use the `@FunctionalInterface` annotation to make sure your interface is OK.

Pre-Defined Functional Interfaces

- Turns out the `java.util.function` package has many pre-defined generic functional interfaces.
- See the API, `svp`.
- The one that matches our check function signature is called `Predicate<T>`. It has the abstract method signature:
  ```java
  boolean test(T)
  ```
- See TestFilter4.java.
- Even better than the last best one!

Method References

- `ArrayList.sort()` accepts a `Comparator<T>` object that can specify the desired algorithm for comparison of objects of type `T`.
- Turns out `Comparator` is a `Functional Interface`, so you can build a lambda function for a comparator.
- See TestSorting.java.

Method References, Cont.

- But suppose our `Person` class has a method that matches the `Comparator` interface:
  ```java
  public static int compareByAge(Person p1, Person p2) {
    return p1.age - p2.age;
  }
  ```
- In this case you can supply a `Method Reference` instead of building the lambda function.
- See TestSortingAgain.java.
Method References, Cont.

db.sort(Person::compareByAge);

- Wow, that was easy! Even easier than a Lambda!
- The method reference must match the functional interface’s specification.
- You can use non-static methods or methods from an instance. If you wish to supply a constructor, use: ClassName::new
- And you can use existing API methods!

Lambda Functions, Cont.

- You can have multiple lines of code in a lambda, but don’t make them too long.
- You can use as many lambdas for arguments as you wish when invoking a function, as long as they each match up to some functional interface.
- In GUI programming the most common event handler interface, EventHandler<ActionEvent> is a functional interface, so lambdas can be used to attach event code to handlers.