CSE216 Programming Abstractions
Recitation 3

Section 1:

Download the CSE216Rec.zip project on your computer and open with NetBeans. We will learn about polymorphism and Java Comparable Interface in Java through various demos.

Polymorphism:

The word polymorphism literally means a state of having many shapes or the capacity to take on different forms. When applied to object oriented programming languages like Java, it describes a languages ability to process objects of various types and classes through a single, uniform interface.

Demo: PolymorphismDemo.java

Polymorphism in Java has two types: Compile time polymorphism (static binding) and Runtime polymorphism (dynamic binding).

Method overloading is an example of static polymorphism.

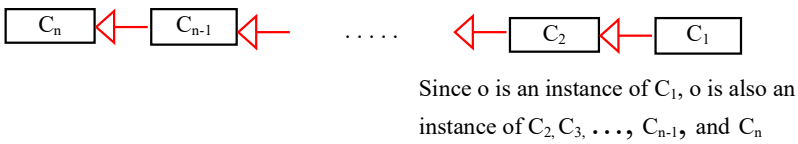
Demo: DisplayOverloading.java

Method overriding is an example of dynamic polymorphism.

Demo: OverridingDemo.java

Dynamic binding: Suppose an object o is an instance of classes C_1, C_2, ..., C_{n-1}, and C_n

- C_1 is a subclass of C_2, C_2 is a subclass of C_3, ..., and C_{n-1} is a subclass of C_n
- C_n is the most general class, and C_1 is the most specific class
- If o invokes a method p, the JVM searches the implementation for the method p in C_1, C_2, ..., C_{n-1} and C_n, in this order, until it is found, the search stops and the first-found implementation is invoked.



Demo: DynamicBinding.java

Generic programming: Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.

Demos: GenericMethodTest.java

BoundedTypes.java

GenericBox.java

Section 2:

Java Comparable interface

By default, a user defined class is not comparable. That is, its objects cant be compared. To make an object comparable, the class must implement the Comparable interface. The Comparable interface has a single method called compareTo() that you need to implement in order to define how an object compares with the supplied object -

```
public interface Comparable<T> {

public int compareTo(T o);

}
```

When you define the compareTo() method in your classes, you need to make sure that the return value of this method is -

- negative, if this object is less than the supplied object.
- zero, if this object is equal to the supplied object.
- positive, if this object is greater than the supplied object.

Many predefined Java classes like String, Date, LocalDate, LocalDateTime etc implement the Comparable interface to define the ordering of their instances.

Study the Employee class which implements Comparable interface as well as ComparableExample class for illustration. You will use this example to solve the following problem:

Suppose there is a class named Product with the following attributes:

`String productCategory`

`int productCost, and`

`String productName.`

Consider a scenario where you have an arraylist of objects of type Product. Write a Java code that uses `java.util.Collections#sort` to sort Product arraylist as follows:

- items will be sorted by `productCost` first,
- if there are multiple items with the same `productCost`, then those will be sorted alphabetically by `productCategory` next, and
- if there are items in the same category and have the same cost, then they will be sorted alphabetically by the `productName` finally.