2021-11-02 -- Scratchpad of CSE213 (Sec-1)
==========

Difference between method overloading and method overriding:

- In overloading, All versions of the method is available for execution for SAME client,
  but in overriding inherited version can never be called for subclass client

- For overloading, ALL versions must have unique parameter list, but in overriding both
  inherited and own version has same parameter list (hence to avoid ambiguity, deactivate
  inherited one)

- Overloaded versions may be written in same class, OR
  some of them can be written in superclass and some of them can be written in subclass.
  BUT, overridden versions will be written: one in superclass, and another in subclass

==========

Relationships among classes:
---

a) Inheritance
    - Extension of preexisting superclass

b) Aggregation
    - A class contains a handle of another preexisting class, as its field
    - Establishes "has-a" relationship
    - Aggregation implies a relationship where the aggregated (which is a Preexisting class's
      handle as a field of new class being defined) class instance can exist independently
      of the aggregating (the new class) class instance.

        Example-1:

        - Toilet is a preexisting class
            public class Toilet {....}

        - Auditorium is the new class being defined
            public class Auditorium{
                private Toilet[] toiletArr;
            }

        - Here Auditorium is aggregating class, and Toilet is aggregated class,
          and Toilet instances exists outside/independent of Auditorium in IUB

```
                Example-2: Totota Corporation
                ----------------------------
                 - Engine is a preexisting class
                        public class Engine {....}

                 - Car is the new class being defined
                        public class Car{
                                private Engine eng;
                        }

                - Here Car is aggregating class, and Engine is
aggregated class,
                  and Engine instances exists outside of Car instance
as SPARE paers in Toyota



c) Composition
        - A class contains a handle of another preexisting class, as
its field
        - Establishes "has-a" relationship
        - Composition implies a relationship where the composed (which
is a Preexisting class's
           handle as a field of new class being defined) class
instance can NOT exist
           independently of the composing (the new class) class
instance.

                Example-1:
                ----------
                 - Engine is a preexisting class
                        public class Engine {....}

                 - Car is the new class being defined
                        public class Car{
                                private Engine eng;
                        }

                - Here Car is composing class, and Engine is composed
class,
                  and Engine instances Cannot exists outside of Car
instance in IUB

d) Association
        - When One class's method interact with another class's method
(using one another),
          but there is no "is-a" / "has-a" relationship, then we call
it association


================================================================
=================

Polimorphism:
-------------
        - Poly: many
```

- Morphing: Smooth transition of image to different form

- Polymorphism in Java is the ability of an object to take many forms.
To simply put, polymorphism in java allows us to perform the same action in many
different ways. It is a feature of the object-oriented programming, which allows a
single task to be performed in different ways (depending on runtime context).

- Inheritance is involved in acheiving polymorphism (as superclass handle can be used
to represent any subclass instances of the inheritance hierarchy)

Example:

Video Game: pubji, Valorent

- Assume, a warfare game has different types of weapons as resources for the player

```
public class Granade{...}
public class Pistol{...}
public class Sword{...}
```

- To preserve the collected weapons, there need to be 3 DIFFERNT ARRAYs
(more dynamic collection alternate of Array) for Granade, Pistol & Sword

```
Granade[] granadeArr;
Pistol[] pistolArr;
Sword[] swordArr;

granadeArr[i] = new Granade();
PistolArr[i] = new Pistol();
swordArr[i] = new Sword();
```

- But if we want to maintain only ONE ARRAY instead of three, then we can
introduce a superclass called Weapon, and make Granade, Pistol & Sword
as subclasses, and then we can use a SINGLE array of Weapon

```
public class Weapon{...}
public class Granade extends Weapon{...}
public class Pistol extends Weapon{...}
public class Sword extends Weapon{...}
Weapon[] weaponArr;
weaponArr[i] = new Granade();
weaponArr[j] = new Pistol();
weaponArr[k] = new Sword();
```

- In Game player's perspective, its not possible to anticipate how many
resources the player can collect at runtime.

Therefore, using array is not a
good option. Hence we can use one of the dynamic collection classes from
library which can grow on demand in phases.

- Let's decide that we are going to use ArrayList class to collect different
types of weapons for the player and demonstrate polymorphic behaviour of a
method (common-method or uncommon-method??? We are going to discover that)

Next class to cover (7-NOV-21):

- Complete sample workflow for a IRAS goal
- Discussion on different UML diagrams which are typically used for system design.
BUT due to time limitation, we will limit ourselves to "class-diagram" only.
You will know more UML diagrams related to system design in "System Analysis and Design" core course in future

- Introduction to online tool "lucidchart" to create out class diagram

- How to use milestone-1 (CRA-report) to construct milestone-2 deliverables:
  - class diagram
  - database design (file system)