CSYE 6200

Concepts of Object Oriented Design

SOLID

Daniel Peters

d.peters@neu.edu

- Lecture **S O L I D**:

- **S**ingle Responsibility Principle:
- **O**pen-Closed Principle:
- **L**iskov Substitution Principle:
- **I**nterface Segregation Principle:
- **D**ependency Inversion Principle:

SOLID

- Acronym coined by Michael Feathers
- Design principles promoted by Robert C. Martin.
  - Benefits:
    - Flexibility
    - Maintainability
    - Understandability
- https://en.wikipedia.org/wiki/SOLID

Single Responsibility Principle:

- Design each class (or module) with ONLY one **Single** responsibility (*purpose* or task);
- Employ Encapsulation
  - 1. All data (supporting *purpose*) class private;
  - 2. Supply public API (supporting *purpose*);

Single Responsibility Principle:

```
public class Person {
        private int age;
        private String name;
        public String toString() {
           return name + “, age “ + age;
        }
}
```

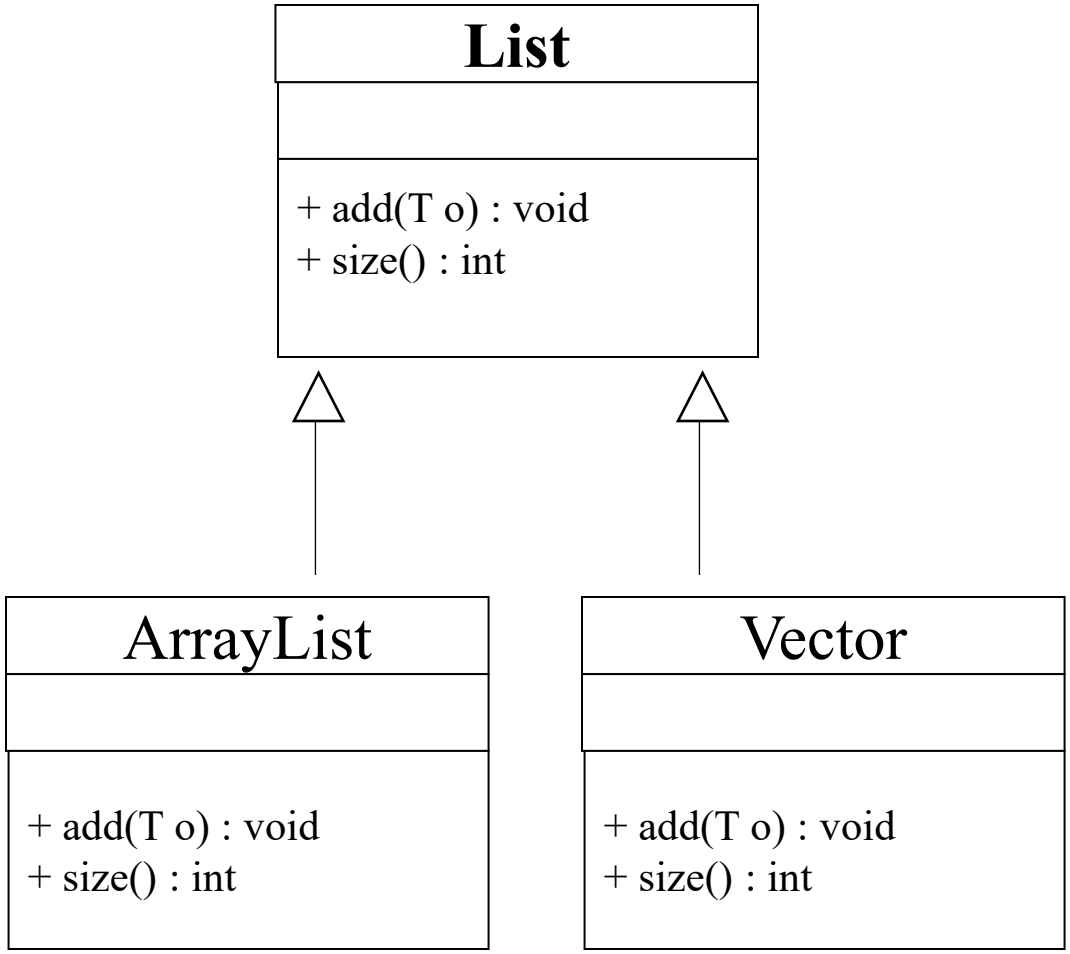
NOT Single Responsibility Principle:

```
public class Person {
        private int age;
        private String name;
        private double gpa;
        private double wage;
        private String toString() {
           return name+“ “+gpa+“ $ “+wage+,/hour
age “+ age +”;
        }
}
```

Open-Closed Principle:

- Each class is **Open** to extension;
  - Use NEW subclasses and Polymorphism
- Each class is **Closed** to modifications;
  - Once tested, use class as a Super class

List Class Diagram



Open-Closed Principle:

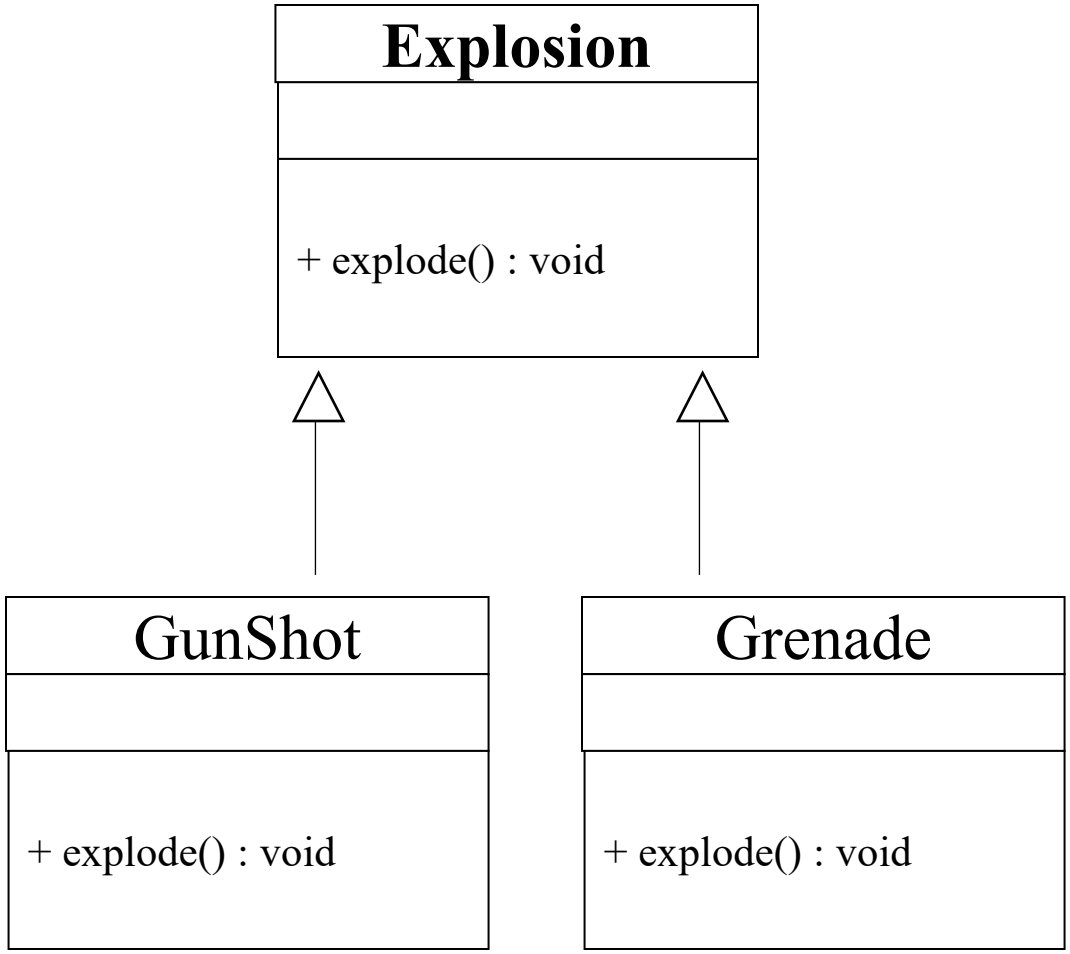
- use derived objects through superclass API

```
List<String> names = new ArrayList<>();
```

- NOT: use derived objects explicitly

```
ArrayList<String> names = new
ArrayList<>();
```

Explosion Class Diagram



Liskov Substitution Principle:

- IS-A Relationship: Any subclass may be **Substituted** for its Super class;
- Run-time Polymorphism;
- Employ Strong subtyping by implementing specific interfaces in subclass;
  - Can be used to differentiate subclasses;

Liskov Substitution Principle:

- use derived objects in substitution of superclass API

```
List<String> names = new ArrayList<>();
List<String> names = new Vector<>();
List<String> names = new LinkedList<>();
```

Interface Segregation Principle:

- No class should depend on any method it does not implement (use);
  - Design Interface:
    - Small granularity
      - fine grained like Salt, not Snowballs
    - Less IS More: Few methods in interface;
    - Very focused and Specific purpose;
    - Class implements multiple interfaces for desired functionality;

Interface Segregation Principle:

http://developer.classpath.org/doc/java/lang/Runnable-source.html

```
public interface Runnable {
        void run();
}
```

http://developer.classpath.org/doc/java/lang/Comparable-source.html

```
public interface Comparable<T> {
        int compareTo(T o);
}
```

Dependency Inversion Principle:

- Loosely coupled Design;
- Depend on abstractions.
  - Functionality Hiding: Subclass is never named
  - Don’t Explicitly call new or static methods
  - Class ExplosionController should use Class Explosion
    - Class GunShot extends Explosion
    - Class Grenade extends Explosion

Dependency Inversion Principle:

- Depend on **List** API as abstraction

List<String> names = **new** ArrayList<>();

  - Use of Factory design pattern abstracts **new** and completes the abstraction of derived classes

- NOT: use derived objects explicitly

ArrayList<String> names = **new** ArrayList<>();

SOLID and OOP

- Loose Coupling is achieved by:
  - Object Oriented Principles (OOP): AIP
    - Abstraction
    - Inheritance
    - Polymorphism
  - SOLID principles: OLD
    - Open-Closed Principle
    - Liskov Substitution Principles
    - Dependency Inversion Principles