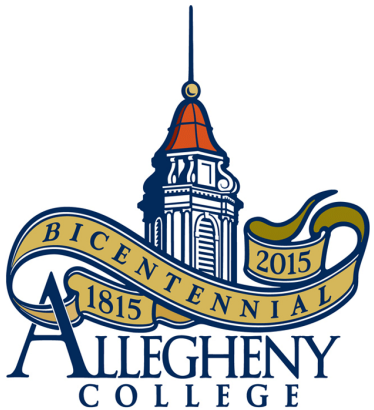
CS101 - Data Abstraction

OOPS - Module1

Aravind Mohan

Allegheny College

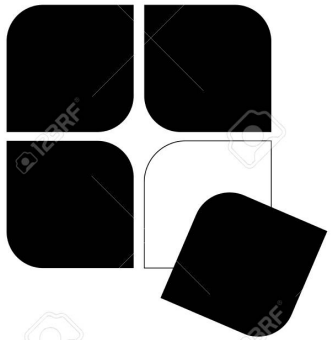
January 28, 2020



Software Goals

Robustness - In addition to producing the correct output for anticipated inputs, we also want the software to handle unexpected inputs not known in advance.

Software Goals



ADAPTABILITY

Adaptability - Software should be able to evolve over time to changing conditions and environment.

Software Goals

Reusability - The same code should be usable as a component in different systems with varying applications.

Software Goals

- **Robustness**
- **Adaptability**
- **Reusability**

Can OOPS support these goals? ...

How does OOPS support these goals?

- Abstraction - Distill a complicated system down into fundamental parts. Specify what each operation does, and how it does it.
- Encapsulation - Different components of a software system should not reveal the internal details of their respective implementations. Data accessed through public interfaces.
- Modularity - Different components of a software system are divided into separate functional units, which later get integrated into a larger software system.

Access Modifiers

- Public - Everything can access. The class, the package, any subclasses, any external classes.
- Protected - Everything can access except for external classes.
- Default / no modifier / ”Package-Private” - Only the class and package can access.
- Private - Only the class can access.

What is a Package? ...

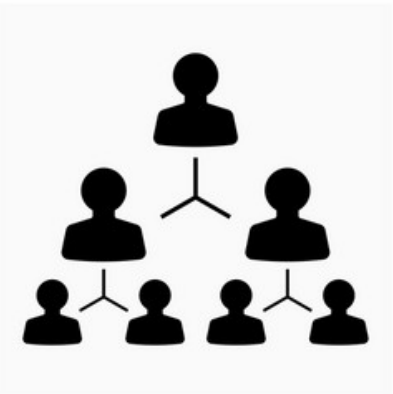
Constructor

- Definition: A special type of method called to create an object.
- This special method is called when a new object is created. Intialization happens in the constructor.

Inheritance

- Definition: A programming technique or mechanism for creating a hierarchy of classes.
- Automatically parent class methods are available in child class.
- Code redundancy is always a big problem.
- Multiple inheritance - A java class can't extend more than one class at a time. Ambiguity problem.

Inheritance



- Is-a relationship
- Has-a relationship (Later: next module)

Is-a relationship - Case 1

```
class p{
  void m1(){
    print("parent");
  }
}
class c extends p{
  void m2(){
    print("child");
  }
}
```

```
p obj = new p();
obj.m1();
obj.m2(); // invalid
```

Is-a relationship - Case 2

```
class p{
  void m1(){
    print("parent");
  }
}
class c extends p{
  void m2(){
    print("child");
  }
}
```

```
c obj = new c();
obj.m1();
obj.m2(); // valid
```

Is-a relationship - Case 3

```
class p{
  void m1(){
    print("parent");
  }
}
class c extends p{
  void m2(){
    print("child");
  }
}
```

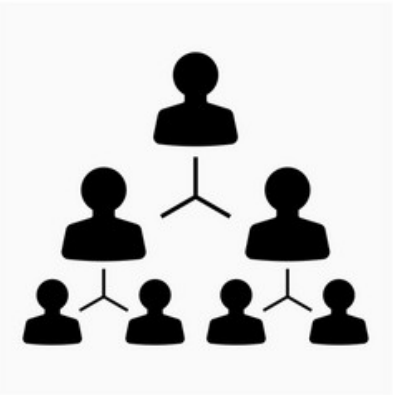
```
p obj = new c();
obj.m1();
obj.m2(); // invalid
```

Is-a relationship - Case 4

```
class p{
  void m1(){
    print("parent");
  }
}
class c extends p{
  void m2(){
    print("child");
  }
}
```

```
c obj = new p(); // invalid
```

Inheritance and Constructor



While creating a child class object, parent constructor will be executed but the parent object will not be created.

Why? Initialization of parent using Super keyword.

Unified Modeling Language (UML)

- Provides a standard way to visualize the design of a system.
- Class Diagram - Describes the structure of a system showing the system's classes, their attributes and operations, and the relationships between them.

  public (+)
  private (-)
  protected (#)
  default ()

Unified Modeling Language (UML)

```
public class Song{
  private String album;
  private String artist;
  protected float duration;
  private String title;
  public void setAlbum(String album){
    this.album = album;
  }
  public String getAlbum(){
    return this.album;
  }
  protected void setDuration(float duration){
    this.duration = duration;
  }
  protected float getDuration(){
    return this.duration;
  }

}
```

Class Diagram

Lets draw the class diagram on the board. Make notes.

Reading Assignment

GT Chapter 2 [2.1,2.2]

Questions?

Please ask if there are any Questions!