CSE 1322
Module 3 – Part 3

Abstract Classes

Encapsulation

- We have learned that it is good practice to bundle up data related to a “thing”:
  - Student -> Name, Age, GPA, etc.
  - Dog -> Name, Weight, Breed, etc.
  - Building -> Units, Floors, Occupancy, etc.
- We used encapsulation to bundle the data related to these “things” into classes.

Encapsulation

- We also learned that through these bundles, we can control the access to this data, hiding the internal details and exposing only the necessary parts (access modifiers).
- Encapsulation deals with the “How” the object work.

Abstraction

- Through abstraction, we focus on the essential behavior of an object.
- We hide the implementation details and complexity.
- With abstraction, our goal is to only show the necessary functionality to the user while hiding its internal work.
- We can implement Abstraction in Java through **Abstract Classes** and **Interfaces**.
- Abstraction deals with the **“What”**.

Abstraction – What

- What the class or object can do?
- What is relevant to the user?
- What behaviors should be exposed?
- What is essential?
- We only expose only the relevant behaviors to the user and hide everything else.

Abstraction – What

- The user should only care about the functionality or purpose rather than the specific implementation

```
String message = "Hello World";

System.out.println(message.length());
```

- Do we care how the **length()** function figure out the number of characters in the string?

Abstract Classes

- An **Abstract Class** is a type of class that cannot be instantiated on its own.
- Think of it as a blueprint for other classes.
- It can contain both **Concrete Methods** and **Abstract Methods**.
- Through Abstract Classes we can provide a **base structure** for subclasses to implement specific details while sharing common functionality.

Abstract Classes – Concrete Methods

- Concrete methods are regular methods where we define its return type, identifier, parameters, **and implementation**.

```
public void printHelloWorld(){
    System.out.println("Hello World!");
}
```

Abstract Classes – Abstract Methods

- Abstract Methods are methods that are declared but its functionality is not defined.
- Think of it as creating the blueprint for a method, specify its return type, identifier, and its parameters.
- While all of those are specified, the implementation is not defined and **must** be defined by any of its **subclasses**.

Abstract Classes – Key Features

- Cannot be instantiated
- Can have both Abstract and Concrete methods
  - Since Abstract Classes can have both, it's a **partial abstraction**.
- Can have fields (attributes)
- Inheritance
  - Since it cannot be instantiated

Abstract Classes – Key Features

- There is a key distinction regarding abstract methods:
  - An **Abstract Class** may contain none or multiple Abstract Methods.
  - **Interfaces** only contain Abstract Methods.

Abstract Classes – Subclasses

- Generally, subclasses of abstract classes will implement **all the abstract methods** inherited.
- If a subclass **partially implements** its inherited abstract methods, it must be an **abstract class.**
  - Yes, subclasses can also be Abstract!
- If a subclass **implements all** its abstract methods, it may be either a **concrete** or **abstract class.**

Abstract Classes – Why?

- Since Abstract Classes can contain Abstract Methods, any subclass will be required to specified the implementation of these type of methods.
- This ensures consistency across related classes.

Abstract Classes – Why?

- Since Abstract Methods must be implemented by each subclass, we can also use them to declare any shared behavior to avoid duplication of similar functions in subclasses.
- We keep reinforcing the concept of **reusable code**.

Abstract Classes – Why?

- Therefore, in the Superclass Abstract Class we focus on the **what**, meaning we define the behaviors.
- While on the Subclasses we define the **how**, or the actual implementation of such behaviors.

Abstract Classes – Syntax

Defining an Abstract Class

```
abstract class Mammal{

}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Defining an Abstract Class

```
abstract class Mammal{
    public double temp;
    public double weight;
    public int IQ;
}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Defining an Abstract Class: Concrete Methods

```java
abstract class Mammal{
    public double temp;
    public double weight;
    public int IQ;

    public Mammal(double temp, double weight, int IQ){
        this.temp = temp;
        this.weight = weight;
        this.IQ = IQ;
    }

    // Concrete Methods
    public void eat(){
        System.out.println("This Mammal is eating");
    }
    public void drink(){
        System.out.println("This Mammal is drinking");
    }
}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Defining an Abstract Class: Abstract Methods

```java
abstract class Mammal{
    public double temp;
    public double weight;
    public int IQ;

    public Mammal(double temp, double weight, int IQ){
        this.temp = temp;
        this.weight = weight;
        this.IQ = IQ;
    }

    // Concrete Methods
    public void eat(){
        System.out.println("This Mammal is eating");
    }
    public void drink(){
        System.out.println("This Mammal is drinking");
    }

    // Abstract Method
    public abstract void talk();
}
```

Week-6/Abstract/HumanExample.java

KENNESAW STATE UNIVERSITY

Abstract Classes – Syntax

Defining an Abstract Class subclass

```
abstract class Primate extends Mammal{
    public String locomotion;
    public Primate(double temp, double weight, int IQ, String locomotion){
        super(temp, weight, IQ);
        this.locomotion = locomotion;
    }
}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Defining a Concrete Class

```
class Human extends Primate{
    public Human(double temp, double weight, int IQ, String locomotion){
        super(temp, weight, IQ, locomotion);
    }
}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Defining a Concrete Class – Implementing the talk() method.

```
class Human extends Primate{
    public Human(double temp, double weight, int IQ, String locomotion){
        super(temp, weight, IQ, locomotion);
    }

    @Override
    public void talk(){
        System.out.println("Hello World!");
    }
}
```

Week-6/Abstract/HumanExample.java

KENNESAW STATE UNIVERSITY

Abstract Classes – Syntax

Implementing in the Driver

```
public class HumanExample {
    public static void main(String[] args) {
        Primate p1 = new Human(37, 80, 130, "Bipedal");

        p1.talk();
    }
}
```

Week-6/Abstract/HumanExample.java

Abstract Classes – Syntax

Implementing in the Driver

```
public class HumanExample {
    public static void main(String[] args) {
        Primate p1 = new Human(37, 80, 130, "Bipedal");

        p1.talk();
    }
}
```

```
Hello World!
```

Week-6/Abstract/HumanExample.java