CAB302 LECTURE NOTES

OOP Part 1 and 2

Week 1 and 2

1 Console input and output

- Standard output is a character stream

```
System.out.println("Hello, World!");
```

- Standard output is a byte stream

```
BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in));
input = stdIn.readLine();
```

- Java 6 introduced class `Console` to facilitate elegant input and output
- To declare a console object:

```
Console terminal = System.console();
```

- Reading from the console:

```
input = terminal.readLine();
```

- Writing to the console:

```
terminal.printf("Hello, World!");
```

2 Classes in Java

- Declare classes with the `class` keyword
- Declare objects with the `new` operator

Types of variables in java classes:

- fields- declared in a class

- local variables- declared within a method
- parameters- passed to a method and part of a method's signature

Within a class the keyword `this` denotes the current object

- `this()` invokes the constructor
- `this.field` accesses an instance field
- `this.method()` calls an instance method

The `static` qualifier means only one copy exists of the field or method

3 Packages in Java

- Java packages are containers for functionally-related classes
- Each package has its own scope and name space
- Classes in different packages can be imported into other classes using the `import` keyword

4 Encapsulation

- A design practice that separates specification (what it does and how to use it) from implementation (actual code)
- In Java, method signatures specify how methods should be called and what they return
- To obey the principles of abstraction and encapsulation of a class, you should:
  - make fields `private`
  - make `accessors` and `mutators` `public` make helper methods `private`

5 Inheritance

- instantiated from from subclasses can do everything that superclass objects can and sometimes more
- They can also override some superclass characteristics
- In Java, inheritance is introduced in classes via the `extends` keyword for inheriting from superclasses
- a superclass' constructor (a subroutine that is called to create an object) is called with `super` and it's methods with `super.method()`

6 Finality

The `final` keyword prevents

- a variable's value from being altered
- a class from being extended
- a method from being overridden

7 Interfaces in Java

- In Java, an interface is a kind of class that contains `abstract` methods only interfaces cannot be instantiated as objects, their abstract methods are incomplete and cannot be executed
- Use keyword `implements` for inheriting from interfaces
- all methods within an interface are `public` by default

Why use interfaces?

- each implementation is very different
- acts as contract- each implementing class HAS to provide implementation
- Multiple inheritance- a java class can only extend one class, but you can implement any number of interfaces

8 Abstract classes in Java

- An abstract class implements some member fields and methods but leaves others abstract
- abstract methods are indicated by the `abstract` keyword
- child class must implement abstract classes but don't have to implement non-abstract classes
- like interfaces, abstract classes cannot be instantiated

Why use Abstract classes?

- code reuse
- polymorphism no need for subclasses to implement non-abstract

9 Polymorphism

- Apply the same operation on different types with a common ancestor in the type hierarchy

10 Enum types

- special kind of class whose fields are `named constants`
- enums have an implicit `values` method- can be used in for loop
- java allows `enum` classes to have other fields and methods, apart from the constants

```
// Enum class example

    public enum Planet{
        MERCURY (3.7),
        VENUS (8.87),
        EARTH (9.799);

        double gravity;
        Planet(double gravity)
        {
            this.gravity = gravity;
        }

        public double getGravity()
        {
            return gravity;
        }

        public static void main(String[] args)
        {
            for (Planet p : Planet.values())
            {
                System.out.println(p.getGravity());
            }
        }
    }
```

11 Other Definitions

- `Accessors` are methods used to return the value of a private field. An accessor is declared as public. The naming scheme of accessors is `getNameOfReturnValue`. The data type of an accessor is the same as their returning private field.
- `Mutator` methods change things. Mutators are declared as public. Mutators do not have a return type, instead they set the value of the private field. They also accept a parameter of the same data type as the private field they are modifying. The naming scheme of mutators is `set nameOfModifiedValue`.

```
    // Accessors and Mutators Example
    public class Cat
    {
        private int Age;
        public int getAge()
```

```
    {
        return this.Age;
    }
    public void set Age(int Age)
    {
        this.Age = Age;
    }
}
// OR
public class Employee
{
    private int number;
    // this is the accessor method
    public int getNumber() {
        return number;
    }
    public void setNumber(int newNumber) {
        number = newNumber;
    }
}
```

- `Abstract` methods are specifications, consisting of a signature but no body

---

```
// Abstract methods Example
public interface GraphicalEntity
{

    public float getArea();

    public float getPerimeter();
}

// INHERITING INTERFACES EXAMPLE
public class Rectangle implements
    GraphicalEntity
    {
        float length;
        float width;

        public Rectangle(float l, float w)
        {
        length = l;
        width = w;
        }

        public float getArea()
        {
        return length * width;
        }

        public float getPerimeter()
```

```
    {
    return length + width * 2;
    }
}
```