Java OOP Concepts Cheat Sheet

Inheritance

- ✓ Inheritance, as name itself suggests, is used to inherit properties from parent class to child class.
- ✓ Using inheritance, you can reuse existing tried and tested code.
- ✓ Using inheritance, you can also add more features to existing class without modifying it by extending it through its subclass.
- ✓ In Java, inheritance is implemented by using extends keyword.
- ✓ An example :

```
class SuperClass
{
    String superClassField = "Super_Class_Field";

    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    String subClassField = "Sub_Class_Field";

    void subClassMethod()
    {
        System.out.println("Sub_Class_Method");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SubClass subClass = new SubClass();

        subClass.subClassMethod();
        System.out.println(subClass.subClassField);

        //SuperClass properties are inherited to SubClass

        subClass.superClassMethod();
        System.out.println(subClass.superClassField);
    }
}
```

Abstraction

- ✓ In computer science terms, abstraction means separating ideas from their actual implementations.
- ✓ Using abstraction, you define only ideas in one class so that those ideas can be implemented by its subclasses according to their requirements.
- ✓ In Java, abstraction is implemented by abstract classes and interfaces.
- ✓ An abstract Class example :

```
abstract class AbstractClass
{
    abstract void anIdea();
}

class SubClassOne extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is implemented according to SubClassOne requirement");
    }
}

class SubClassTwo extends AbstractClass
{
    @Override
    void anIdea()
    {
        System.out.println("An idea is implemented according to SubClassTwo requirement");
    }
}
```

✓ An interface example :

```
interface Interface
{
    void anIdea();
}

class ClassOne implements Interface
{
    @Override
    public void anIdea()
    {
        System.out.println("An idea is implemented according to ClassOne requirement");
    }
}

class ClassTwo implements Interface
{
    @Override
    public void anIdea()
    {
        System.out.println("An idea is implemented according to ClassTwo requirement");
    }
}
```

Polymorphism

- ✓ Poly means many and morphs means forms. So, anything which has multiple forms is called as polymorphism.
- ✓ In computer science terms, any entity like operator or method or constructor which takes many forms and can be used for multiple tasks is called as polymorphism.
- ✓ For example, '+' operator can be used for addition of two numbers as well as for concatenation of two strings.
- ✓ In Java, there are two types of polymorphism - static polymorphism and dynamic polymorphism.
- ✓ Any entity which shows polymorphism during compilation is called static polymorphism.
- ✓ Operator overloading, method overloading and constructor overloading are best examples of static polymorphism.

```
class AnyClass
{
    int i;
    String s;

    //Constructor Overloading

    public AnyClass()
    {
        this.i = 1;
        this.s = "";
    }

    public AnyClass(int i, String s)
    {
        this.i = i;
        this.s = s;
    }

    //Method Overloading

    void anyMethod(int i)
    {
        System.out.println(i+this.i);    //Here, '+' is used to add two numbers
    }

    void anyMethod(String s)
    {
        System.out.println(s+this.s);    //Here, '+' is used to concatenate two strings
    }
}
```

- ✓ Any entity which shows polymorphism at run time is called as dynamic polymorphism.
- ✓ Method overriding is the best example of dynamic polymorphism.

```
class SuperClass
{
    void superClassMethod()
    {
        System.out.println("Super_Class_Method");
    }
}

class SubClass extends SuperClass
{
    @Override
    void superClassMethod()
    {
System.out.println("Super_Class_Method_Is_Overridden");
    }
}

public class JavaOOPConcepts
{
    public static void main(String[] args)
    {
        SuperClass superClass = new SuperClass();

        superClass.superClassMethod();
//Output : Super_Class_Method

        superClass = new SubClass();

        superClass.superClassMethod();
//Output : Super_Class_Method_Is_Overridden
    }
}
```

Encapsulation

- ✓ Bundling of data and operations to be performed on that data into single unit is called as encapsulation.
- ✓ Encapsulation in Java can be achieved by including both variables (data) and methods (operations) which act upon those variables into a single unit called class.
- ✓ Encapsulation is often used to hide important information from outside the world. It is called data hiding. This can be achieved by declaring all important variables as private and providing public getter and setter methods.

```
class Customer
{
    private int custID;
    private String name;
    private String address;

    //Getter and setter for custID

    public int getCustID()
    {
        return custID;
    }

    public void setCustID(int custID)
    {
        this.custID = custID;
    }

    //Getter and setter for name

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    //Getter and setter for address

    public String getAddress()
    {
        return address;
    }

    public void setAddress(String address)
    {
        this.address = address;
    }
}
```