Experiment No._____ **Date___/___/2020**

TITLE OF EXPERIMENT: - A program Inheritance in Java

DIVISION:_____ **BRANCH:** _____

BATCH:_____ **ROLL NO.:** _____

PERFORMED ON DATE: _____

SIGNATURE OF TEACHING STAFF:

EXPERIMENT NO. 7

Aim: Write a program in Java to create a player class. Inherit the classes Cricket_player, Football_player and Hockey_player from player class.

Objectives: To learn the use of inheritance and constructor in java.

Software:

1. Eclipse
2. JDK 16

Theory:

Theory:

Inheritance: Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications. Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass. It inherits all of the instance variables and methods defined by the superclass and adds its own, unique elements.

Important Points

1. Code reuse is the most important benefit of inheritance because subclasses inherits the variables and methods of superclass.
2. Private members of superclass are not directly accessible to subclass. As in this example, Animal variable noOfLegs is not accessible to Cat class but it can be indirectly accessible via getter and setter methods.
3. Superclass members with default access is accessible to subclass ONLY if they are in same package.
4. Superclass constructors are not inherited by subclass.
5. If superclass doesn't have default constructor, then subclass also needs to have an explicit constructor defined. Else it will throw compile time exception. In the

subclass constructor, call to superclass constructor is mandatory in this case and it should be the first statement in the subclass constructor.

6. **Java doesn't support multiple inheritance**, a subclass can extends only one class. Animal class is implicitly extending Object class and Cat is extending Animal class but due to java inheritance transitive nature, Cat class also extends Object class.
7. We can create an instance of subclass and then assign it to superclass variable, this is called **upcasting**. Below is a simple example of upcasting:

```
Cat c = new Cat(); //subclass instance
Animal a = c; //upcasting, it's fine since Cat is also an Animal
```

8. When an instance of Superclass is assigned to a Subclass variable, then it's called **downcasting**. We need to explicitly cast this to Subclass. For example;

```
Cat c = new Cat();
Animal a = c;
Cat c1 = (Cat) a; //explicit casting, works fine because "c" is actually of type Cat
```

Note that Compiler won't complain even if we are doing it wrong, because of explicit casting. Below are some of the cases where it will throw **ClassCastException** at runtime.

```
Dog d = new Dog();
Animal a = d;
Cat c1 = (Cat) a; //ClassCastException at runtime

Animal a1 = new Animal();
Cat c2 = (Cat) a1; //ClassCastException because a1 is actually of type Animal at runtime
```

9. We can override the method of Superclass in the Subclass. However we should always annotate overridden method with at Override annotation. The compiler will know that we are overriding a method and if something changes in the superclass method, we will get a compile-time error rather than getting unwanted results at the runtime.
10. We can call the superclass methods and access superclass variables using **super** keyword. It comes handy when we have the same name variable/method in the subclass but we want to access the superclass variable/method. This is also used when Constructors are defined in the superclass and subclass and we have to explicitly call the superclass constructor.
11. We can't extend Final classes in java.
12. If you are not going to use Superclass in the code i.e your Superclass is just a base to keep reusable code then you can keep them as Abstract class to avoid unnecessary instantiation by client classes. It will also restrict the instance creation of base class.

Program:

This program shows a class hierarchy of the various players in a team like Cricket player, Football player and Hockey player

```
class Player
{
  String name;
  int age;
  Player(String n,int a)
  { name=n; age=a; }
  void show()
  {
     System.out.println("Player name: "+name);
     System.out.println("Age: "+age);
  }
}
class cricket_player extends Player
{
  String type;
  cricket_player(String n,String t,int a)
  {
     super(n,a);
     type=t;
  }
  public void show()
  {
     super.show();
     System.out.println("Player type : "+type);
  }
}
class football_player extends Player
  {
     String type;
     football_player(String n,String t,int a)
  {
     super(n,a);
     type=t;
  }
   public void show()
   {
     super.show();
     System.out.println("Player type : "+type);
   }
}
class hockey_player extends Player
```

```
{
   String type;
   hockey_player(String n,String t,int a)
   {
      super(n,a);
      type=t;
   }
   public void show()
   {
      super.show();
      System.out.println("Player type : "+type);
   }
}
public class Demo
{
   public static void main(String args[])
   {
     cricket_player c=new cricket_player("Sachin","Cricket",20);
     football_player f=new football_player("Namrata","Football",21);
     hockey_player h=new hockey_player("Mandeep","Hockey",22);
     c.show();
     f.show();
     h.show();
   }
}
```

Output:

```
Player name: Sachin
Age: 20
Player type : Cricket
Player name: Namrata
Age: 21
Player type : Football
Player name: Mandeep
Age: 22
Player type : Hockey
```

Conclusion:

Screenshot's of Program and Result: