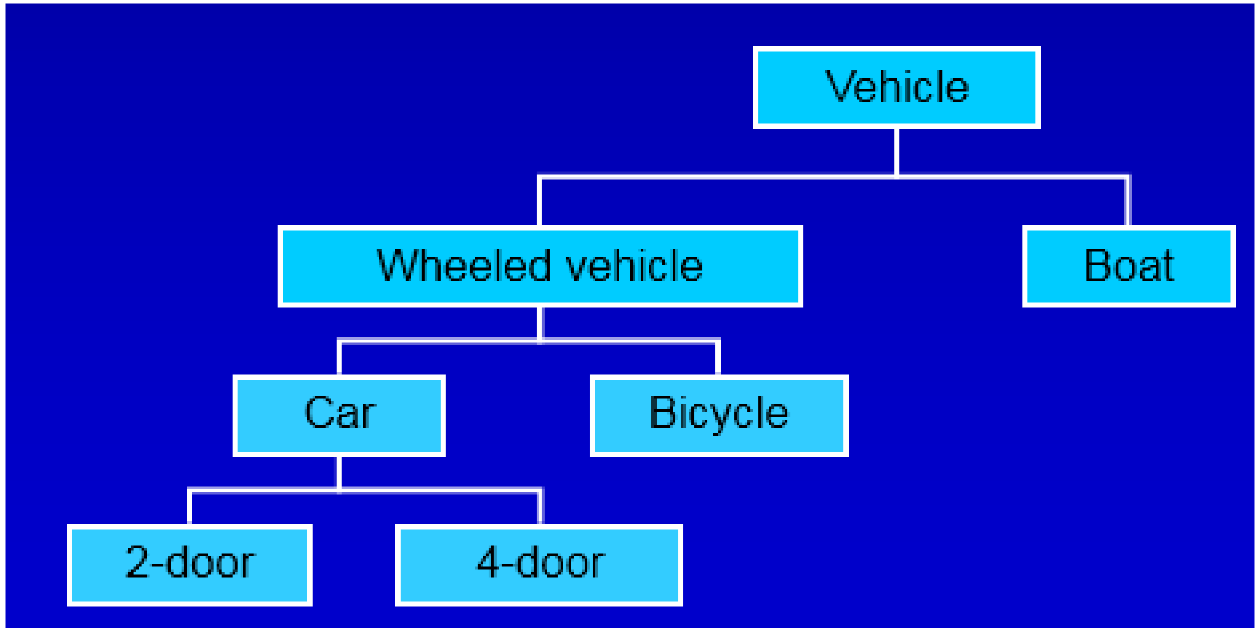
# INHERITANCE

# Inheritance

- Inheritance (pewarisan) adalah mekanisme di OOP yang memungkinkan class baru dibuat berdasarkan class yang sudah ada sebelumnya.
- Mendesain konsep dalam hirarki inheritance



- Konsep di level atas lebih general
  (superclass, baseclass)

- Konsep di level lebih bawah lebih spesifik
  (subclass, derivedclass)

## Inheritance

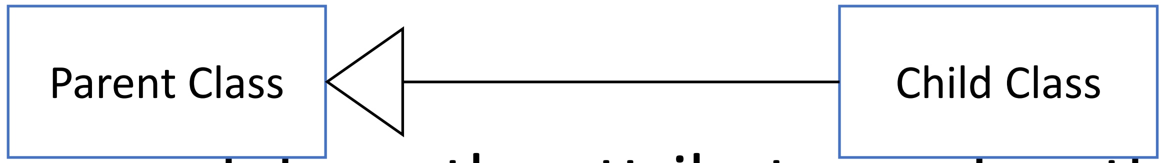
- The ability of a java class to ‘pass down’ all or some of their fields, attributes, and methods to another class that will be referred to as their child class
- Child class will be able to recognize the inherited fields and methods, and can use it without declaring or defining again

## Inheritance

- Inheritance is an implementation of generalization-specialization class relationship or "is-a" relationship
- In short : it’s the mechanism by which one class acquires the properties of another class

# Inheritance

- Denoted by arrows (hollow and triangular head)



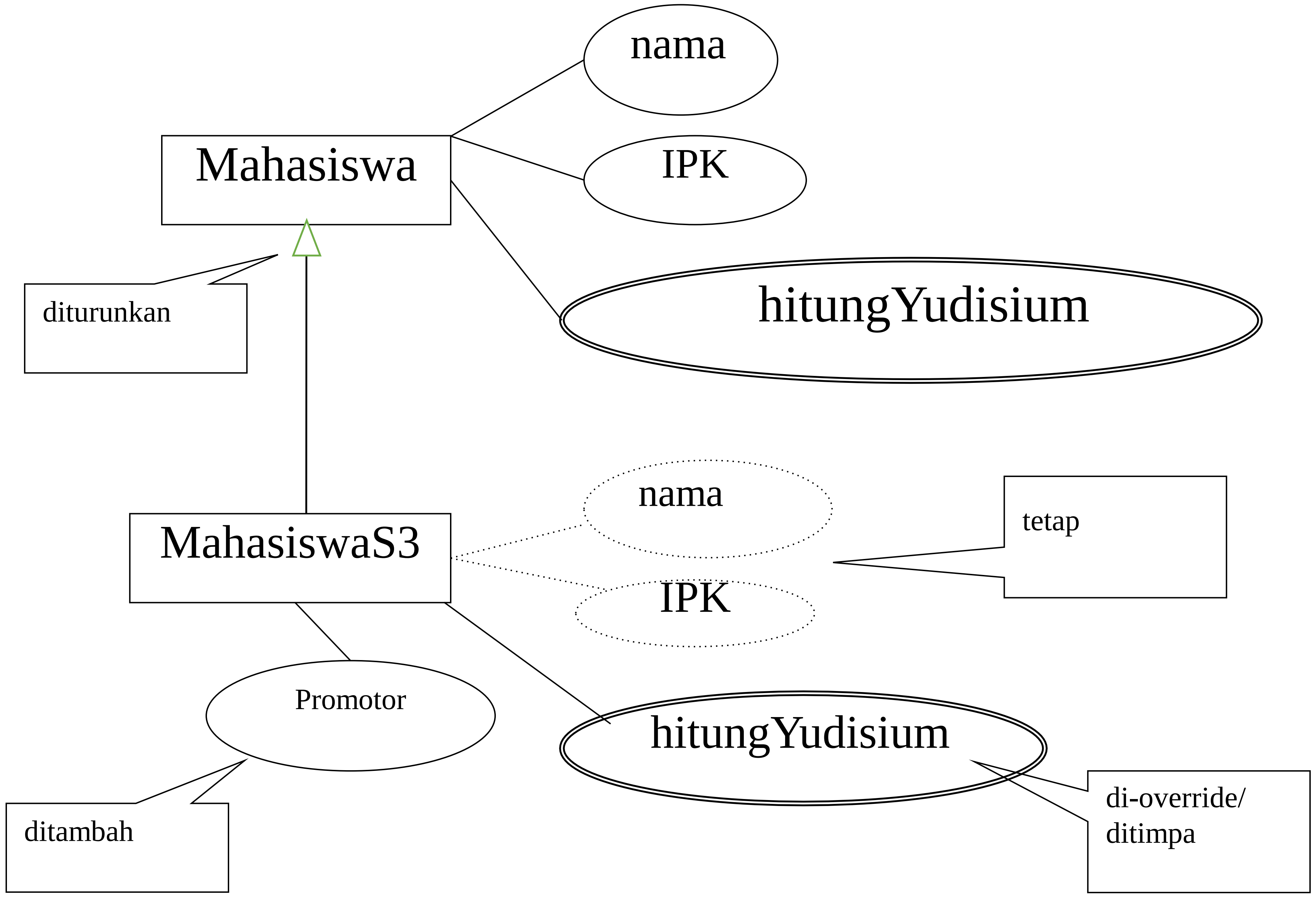
- The classes that passed down the attributes and methods are called
  - parent class, super class, or base class
- The classes that inherit the attributes and methods are called
  - Child class, subclass, or derived class

## Inheritance: Manfaat

- *Reuse* atribut dan method dari superclass (parent)
- *Extend* superclass (parent) dengan menambahkan atribut dan method baru
- *Modify* superclass (parent) dengan overloading method dengan implementasi sendiri

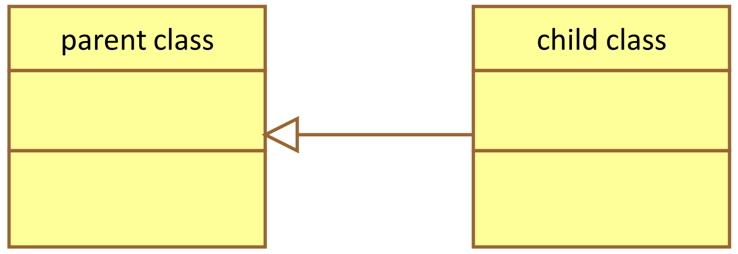
## Inheritance: Contoh

- Misalkan kelas MahasiswaS3 diturunkan dari kelas Mahasiswa.
  Class MahasiswaS3 yang baru ini akan mewarisi atribut dan method milik Mahasiswa, tapi juga bisa menambahkan atribut atau method yang baru.
- Class Mahasiswa akan disebut sebagai **superclass** dan class MahasiswaS3 adalah **subclass**.

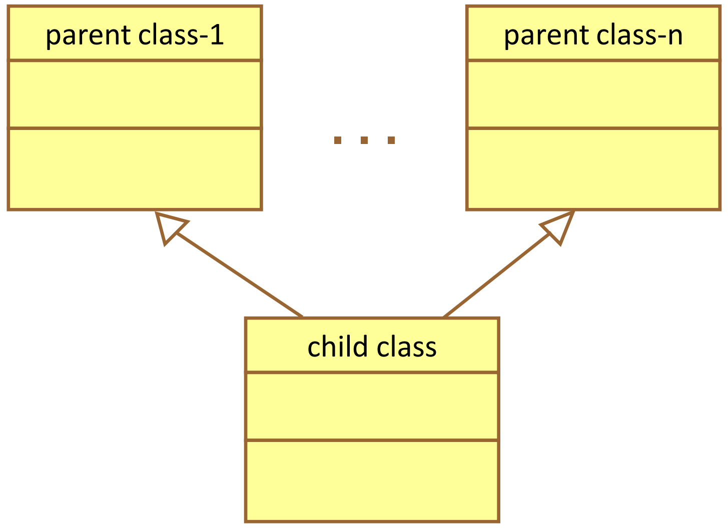

nama
Mahasiswa
IPK
diturunkan
hitungYudisium
nama
tetap
MahasiswaS3
IPK
Promotor
hitungYudisium
ditambah
di-override/
ditimpa

# Types of Inheritance

- Single Inheritance
  - Child class inherit from 1 parent class



- Multiple Inheritance
  - Child class inherit from many parent classes



## Java Class Inheritance

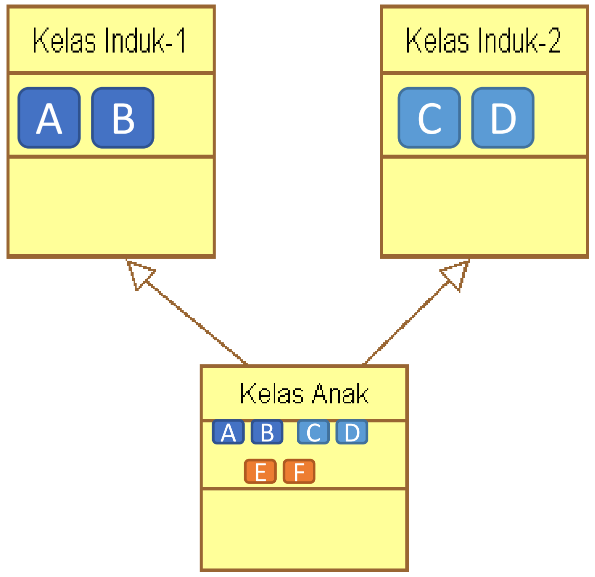
- Single inheritance only
- Using keyword extends

```
class ChildClass extends ParentClass{
  // child class attributes
  // child class methods
}
```

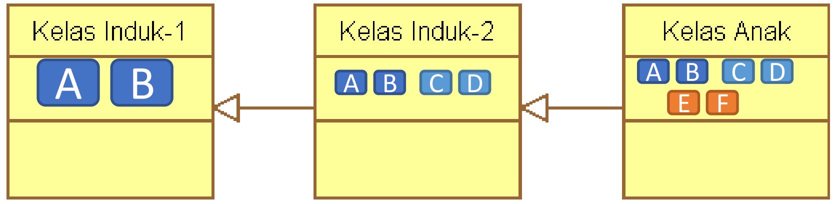
- All non-private attributes and methods are passed on
- To access the properties of parent
  - Use "super" keyword

# Multiple Inheritance in Java

- Multiple Inheritance is handled using staged single inheritance



Base Multiple Inheritance



Multiple Inheritance using Staged Single Inheritance

# Example - Method Inheritance

```
public class Parent{
     public String methodParent(){
        return "this is method Parent";
     }
}
```

```
public class Child extends Parent{
   public String methodChild(){
       return "this is method Child";
   }
}
```

```
public class Driver{
   public static void main(String args[]){
       Parent p = new Parent();
       Child c = new Child();

       System.out.println(p.methodParent());
       System.out.println(c.methodParent());

       System.out.println(c.methodChild());
       System.out.println(p.methodChild());
   }
}
```

```
> This is method Parent
> This is method Parent

> This is method Child
> error: cannot find symbol
```

Child can access public method of Parent, but not vice versa

# Example – Variable Inheritance

```
public class Parent{
        public int publicInt;
        private int privateInt;
        int defaultInt;
}
```

```
public class Child extends Parent{
   public void methodChild(){
      publicInt = 10;
      privateInt = 20;
      defaultInt = 30;
   }
}
```

error: privateInt has private access only can be accessed by itself

## Example

```java
public class Parent{
  public int publicInt;
  private int privateInt;
  int defaultInt;

  public void setPrivateInt(int privateInt){
        this.privateInt = privateInt;
  }
}
```

```java
public class Child extends Parent{
   public void methodChild(){
      publicInt = 10;
      setPrivateInt(20);
      defaultInt = 30;
   }
}
```

privateInt can be accessed by parent method

# Example

```
package test;
public class Parent{
  public int publicInt;
  private int privateInt;
  int defaultInt;

  public void setPrivateInt(int privateInt){
        this.privateInt = privateInt;
  }
}
```

```
import test.Parent;
public class Child extends Parent{
   public void methodChild(){
      publicInt = 10;
      setPrivateInt(20);
      defaultInt = 30;
   }
}
```

error: defaultInt is not public in Parent; cannot be accessed from outside package

## Access Modifier Protected in inheritance

- All class in the same package
- All subclass (child class) even in different package

shutterstock

## Example

```java
package test;
public class Parent{
  public int publicInt;
  private int privateInt;
  int defaultInt;
  protected int protectedInt;

  public void setPrivateInt(int privateInt){
        this.privateInt = privateInt;
  }
}
```

```java
import test.Parent;
public class Child extends Parent{
   public void methodChild(){
      publicInt = 10;
      setPrivateInt(20);
      protectedInt = 30;
   }
}
```

```java
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    c.methodChild();
  }
}
```

Modification to protectedInt done by Child class

## Example

```
package test;
public class Parent{
  public int publicInt;
  private int privateInt;
  int defaultInt;
  protected int protectedInt;

  public void setPrivateInt(int privateInt){
        this.privateInt = privateInt;
  }
}
```

```
import test.Parent;
public class Child extends Parent{

}
```

```
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    c.publicInt = 10;
    c.setPrivateInt(20);
    c.protectedInt = 40;
  }
}
```

error: protectedInt has protected access in parent

(protectedInt is owned by Parent in different package)

## Overriding

- Redefining field or method inherited by Parent class in Child class (Polymorphism)
- subclass can implement a parent class method more specifically based on its requirement.
- When overriding a field or method, the Child class will have access to both the original parent field/method and the new redefined child field/method
- Call parent’s field/method using keyword "super"
    - Keyword "super" only available inside the child class
    - Super is like “this” in a class, but it only use for child

## Example

```
public class Parent{
  public String toString(){
        return "this is method 
Parent";
  }
}
```

```
public class Child extends Parent{


}
```

```
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    Parent p = new Parent();

    System.out.println(p.toString());
    System.out.println(c.toString());
  }
}
```

```
> this is method Parent
> this is method Parent
```

## Example

```java
public class Parent{
  public String toString(){
        return "this is method
Parent";
  }
}
```

```java
public class Child extends Parent{
  public String toString(){
    return "this method overridden
         by Child";
  }
}
```

```java
public class Driver{
  public static void main(String args[]){
   Child c = new Child();
   Parent p = new Parent();

   System.out.println(p.toString());
   System.out.println(c.toString());
  }
}
```

```
> this is method Parent
> this method overridden by Child
```

# Example

```java
public class Parent{
   public String toString(){
         return "this is method 
Parent";
   }
}
```

```java
public class Child extends Parent{
   public String toString(){
     return "this method overridden 
          by Child";
   }
   public String toStringParent(){
     return super.toString();
   }
}
```

```java
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    Parent p = new Parent();

    System.out.println(p.toString());
    System.out.println(c.toString());
    System.out.println(c.toStringParent());
  }
}
```

```
> this is method Parent
> this method overridden by Child
> this is method Parent
```

## Example

```
public class Parent{
  protected int number;

}
```

```
public class Child extends Parent{
  private int number;

  public void methodChild(){
     number = 10;
     super.number = 20;

     System.out.println(number);
     System.out.println(super.number);
  }
}
```

```
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    c.methodChild();
  }
}
```

```
> 10
> 20
```

# Example

```
public class Parent{
  protected int number;

  public void setNumber(int number){
          this.number = number;
  }

  public int getNumber(){
          return number;
  }
}
```

```
public class Child extends Parent{
  private int number;

  public void methodChild(){
     number = 10;
     super.number = 20;
  }
}
```

```
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    c.methodChild();

    System.out.println(c.getNumber());
  }
}
```

```
> 20
```

setNumber() and getNumber() is owned by parent, accessing number value from parent (super)

# Example

```
public class Parent{
  protected int number;

  public void setNumber(int number){
        this.number = number;
  }

  public int getNumber(){
        return number;
  }
}
```

```
public class Child extends Parent{
  private int number;

  public void methodChild(){
     number = 10;
     super.number = 20;
  }
  public int getNumber(){
     return number;
  }
}
```

```
public class Driver{
  public static void main(String args[]){
    Child c = new Child();
    c.methodChild();

    System.out.println(c.getNumber());
  }
}
```

```
> 10
```

If it's overridden, the method will access child field first