Inheritance

Agenda

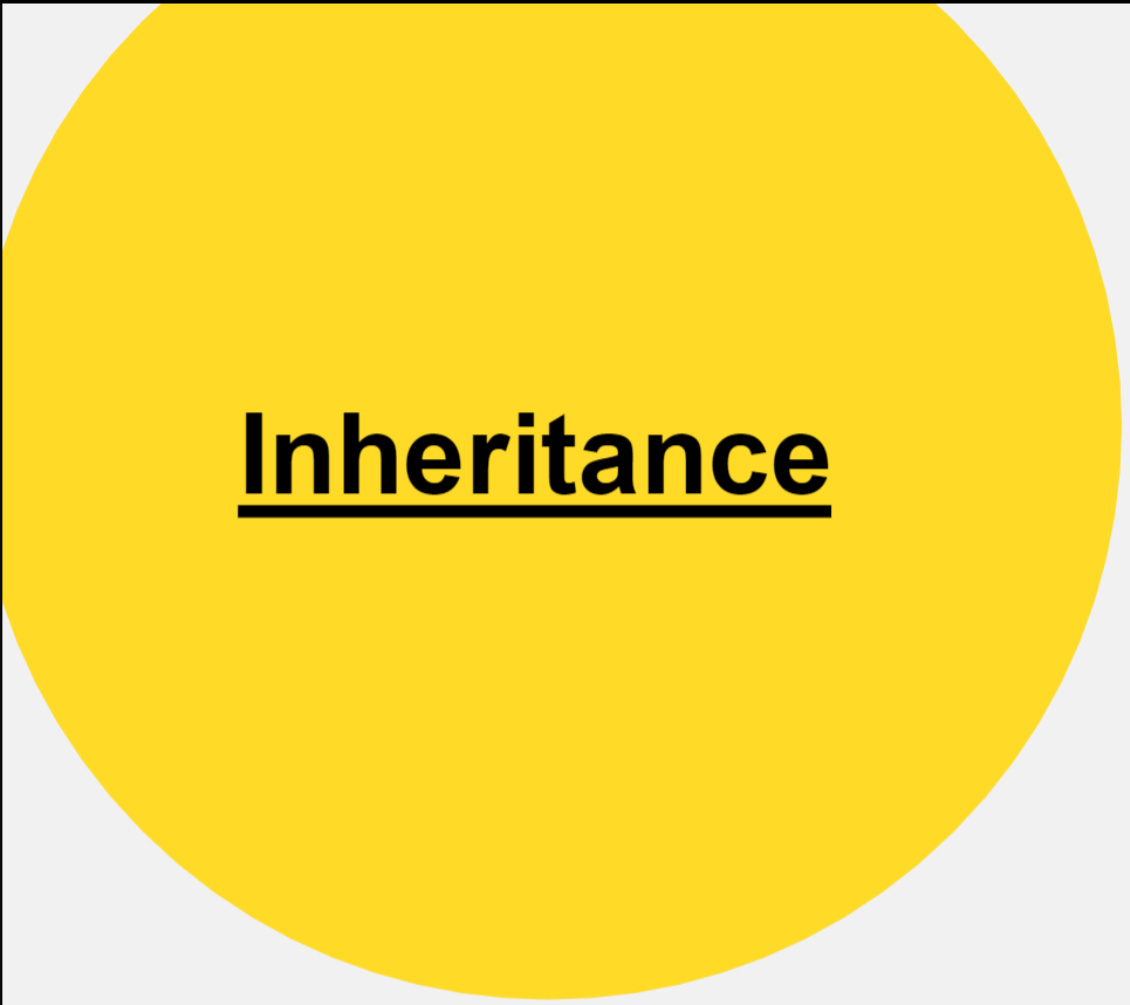
1. Inheritance
1. Multilevel Hierarchy

Objectives

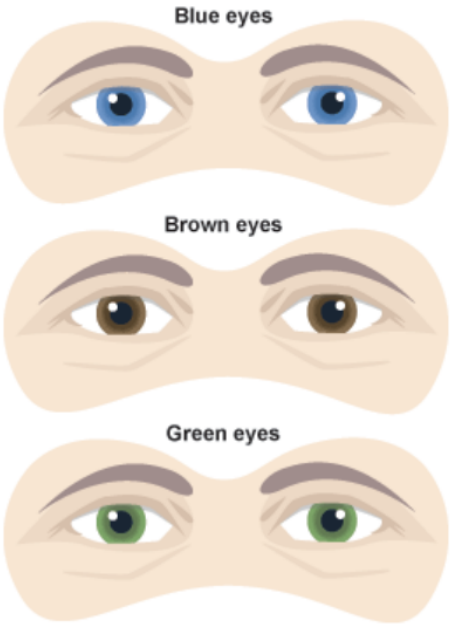
At the end of this module, you will be able to:

- Describe Java's inheritance model and its language syntax
- Describe the usage of the keyword **super**
- Define a multilevel hierarchy
- Describe method overriding
- Describe dynamic method dispatch, or runtime polymorphism
- Understand the use of instanceof operator
- Get basic information about garbage collection
- Define finalize method

Inheritance



Inheritance in real world



Have you seen some people who have BLUE EYES?

Some people have BLUE EYE: How it is possible?

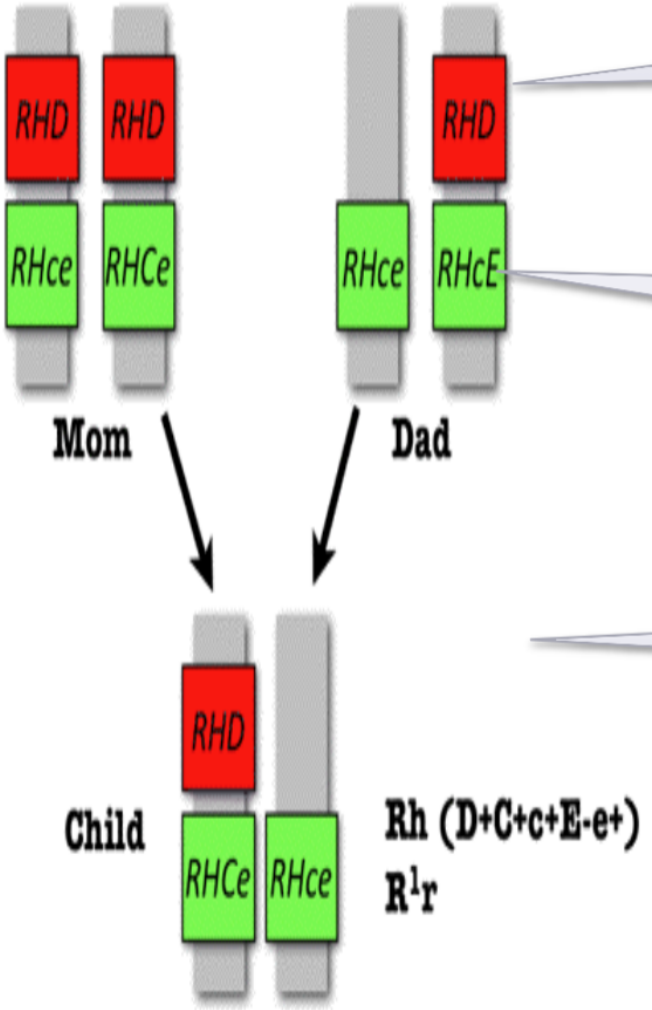
Some people have BROWN EYE: How it is possible?

Some people have GREEN EYE: How it is possible?

Generalization/ Specialization

- In keeping with Java terminology, a class that is inherited is referred to as a superclass
- The class that does the inheriting is referred to as the subclass
- Each instance of a subclass includes all the members of the superclass
- The subclass inherits all the properties of its superclass

Inheritance in real world (Contd.).



In real life, what is meant by inheritance?

What you inherit from your parent? What is your own behavior which is not found in your parent?

Child inherits some properties from its parent. Apart from that, child has its own properties….

Inheritance

- Inheritance is one of the cornerstones of OOP because it allows for the creation of hierarchical classifications
- Using inheritance, you can create a general class at the top
- This class may then be inherited by other, more specific classes
- Each of these classes will add only those attributes and behaviors that are unique to it

IS-A relationship: Manager IS-A Employee

4 Employees of a department

Their Manager

HAS-A relationship

- ***HAS-A*** relationship is expressed with containership
- Containership simply means using instance variables that refer to other objects
- Example:
- The class House will have an instance variable which refers to a Kitchen object
  - It means that, House HAS-A Kitchen
  - Note that, something like Kitchen HAS-A House is not valid in this context

HAS-A relationship (Contd.).

- Let us take one personal computer.
- It has a monitor, CPUbox, keyboard and mouse, etc.
- Technically we can say that,
  - Personal Computer class HAS-A monitor.
  - Personal Computer class HAS-A CPUbox
  - Personal Computer class HAS-A keyboard.
  - Personal Computer class HAS-A mouse.
  - The most important point is : the 4 independent components like monitor, keyboard, CPUbox and mouse cannot function separately on its own.
  - But, by combining them, we are creating a new type of useful class called Personal Computer.

Java's Inheritance Model

- Java uses the single inheritance model
- In single inheritance, a subclass can inherit from **one (and only one)** superclass

Code Syntax for Inheritance:

```
class derived-class-name extends base-class-name
  {
// code goes here
}
```

Inheritance – A Simple Example

```
class A{
  int m, n;
  void display1( ){
    System.out.println("m and n are:"+m+" "+n);
  }
}
class B extends A{
  int c;
  void display2( ){
    System.out.println("c :" + c);
  }
  void sum(){
    System.out.println("m+n+c = " + (m+n+c));
  }
}
```

Inheritance – A Simple Example (Contd.).

```
class InheritanceDemo{
  public static void main(String args[ ]){

    A s1 = new A();    // creating objects
    B s2 = new B();
    s1.m = 10; s1.n = 20;

    System.out.println("State of object A:");
    s1.display1();
    s2.m = 7; s2.n = 8; s2.c = 9;
    System.out.println("State of object B:");
    s2.display1();
    s2.display2();
    System.out.println("sum of m, n and c in object B is:");
    s2.sum();
  }
}
```

Accessing Superclass Members from a Subclass Object

- A subclass includes all of the members of its superclass

- But, it cannot directly access those members of the super class that have been declared as **private**.

```
class A{
  int money;
  private int pocketMoney;

  void fill (int money, int pocketMoney)
  {
    this.money = money;
    this.pocketMoney = pocketMoney;
  }
}
```

Accessing Superclass Members from a Subclass Object (Contd.).

Will this compile now?

```
class B extends A{
  int total;
  void sum( ){
    total = money + pocketMoney;
  }  }
class AccessDemo
{
  public static void main(String args[ ])
  {
    B subob = new B();
    subob.fill(10,12);
    subob.sum();
    System.out.println("Total: " + subob.total);
  }
}
```

A Possible Solution To The Program

```
class A{
  int money;
  private int pocketMoney;
  void fill(int money, int pocketMoney)
  {
    this.money = money;
    this.pocketMoney = pocketMoney;
  }
  public int getPocketMoney(){
    return pocketMoney;
  }
}
```

A Possible Solution To The Program (Contd.).

```
class B extends A{
  int total;
  void sum( ) {
    total = money + getPocketMoney();     }
}
class AccessDemo {
  public static void main(String args[ ]) {
        B subob = new B();
        subob.fill(10,12);
        subob.sum();
        System.out.println("Total: " + subob.total);
  }
}
```

Will this compile now?

Using super

- The creation and initialization of the superclass object is a prerequisite to the creation of the subclass object.
- When a subclass object is created,
  - It creates the superclass object
  - Invokes the relevant superclass constructor.
    - The initialized superclass attributes are then inherited by the subclass object
  - finally followed by the creation of the subclass object
    - initialization of its own attributes through a relevant constructor subclass

Using super (Contd.).

- The constructors of the superclass are never inherited by the subclass
- This is the only exception to the rule that a subclass inherits all the properties of its superclass