Methods

s20

- Similar to procedures
- Allows the programmer to divide the program in modules
- Diminish complexity for
  - Writing
  - Reading
  - Maintaining
    - Changing
    - Extending

- Static
  - Nested Classes
  - Blocks within a class
  - Methods
  - Variables: local and class variables

- Blocks within a class
  - A class can contain various initialisation blocks
  - Those blocks are initialisatied when a class is created
  - Each static block runs in order of appearance

- Example Static block
  - Modulo 2, Pag 13

- Static method or method class
  - Can be invoked without using an object of the class
  - Example
    - Math.sqrt(900.0)

Methods

- Setters
- Getters
- Behaviour

- Why the method main is declared as static?

- A method includes
  - Parameter types
  - Return type
  - Code of the method

```
type methodName(type1 param1, type2 param2,...) {
   code of the method
}
```

- Parameters
  - Are passed by value
  - Only objects are passed by reference

- Return Type
  - If the method do not return any value, it is declared as void
  - If the method returns a value, it uses the construct "return"

- Variable arguments
  - A method can have normal parameters along with a variable length-parameter
  - A variable lenght-parameter involves zero or more parameters
  - The variable length-parameter must be the last parameter
    - It is implicitly declared as an array of the defined type

- Variable arguments
  - No normal parameters involved

```
int aMethod(int ... v) {}
```

  - Normal parameters involved

```
int aMethod(int a, int b, double c, int ... v) {}
```

```
public class MainClass {

  // vaTest() now uses a vararg.
  public static void vaTest(int... v) {
    System.out.print("Number of args: " + v.length + " Contents: ");
    for (int x : v)
      System.out.print(x + " ");

    System.out.println();
  }

  public static void main(String args[]) {
    vaTest(10); // 1 arg
    vaTest(1, 2, 3); // 3 args
    vaTest(); // no args
  }
}
```

Output:
```
Number of args: 1 Contents: 10
Number of args: 3 Contents: 1 2 3
Number of args: 0 Contents:
```

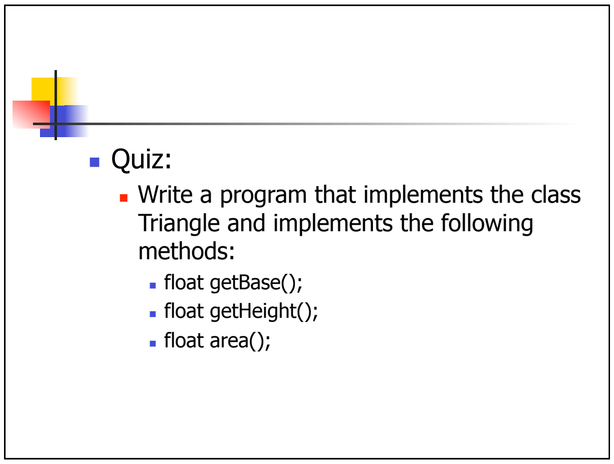
- Local variables take precedence over instance and class variables
- The "this" operator can be used to refer to the instance and class variables

```
Class Foo {
   int size;
   String name;
   Foo(String name, int size) {
      this.name = name;
      this.size = size;
   }
}
```

- We can have several instances

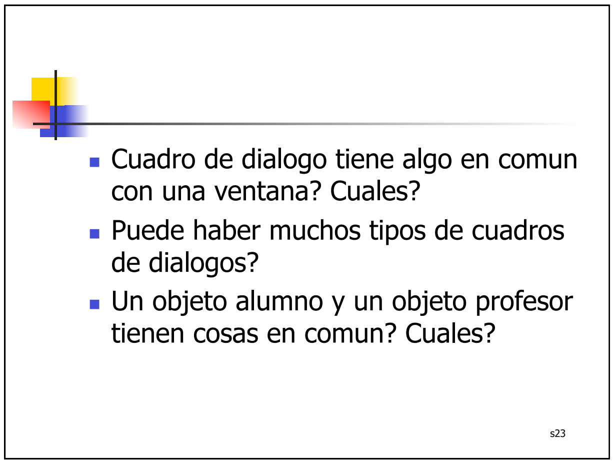
```
int main () {
      Rectangle rect = new Rectalgle();
      Rectangle rectb = new Rectalgle();
      rect.set_values (3,4);
      rectb.set_values (5,6);
      cout << "rect area: " << rect.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
}
```

s18

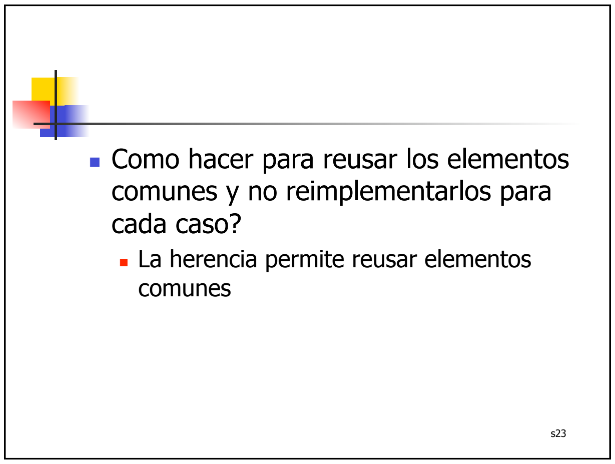


- Quiz:
  - Write a program that implements the class Triangle and implements the following methods:
    - float getBase();
    - float getHeight();
    - float area();

Inheritance



- Cuadro de dialogo tiene algo en comun con una ventana? Cuales?
- Puede haber muchos tipos de cuadros de dialogos?
- Un objeto alumno y un objeto profesor tienen cosas en comun? Cuales?

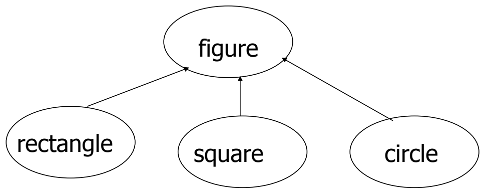


- Como hacer para reusar los elementos comunes y no reimplementarlos para cada caso?
  - La herencia permite reusar elementos comunes

- Base class or super class
- Derived class, subclass or child clas
- Inheritance hierarchy
- The subclass and the super class have a "is-a" relationship
- Syntax

```
class class_name extends base_class_name {
   members;
}
```

Ejemplo



- Types of inheritance
  - Single
  - Multiple
- Java only supports single inheritance

- Access Rights
  - Public
  - Protected
  - Private
  - Default

- Class access rights
  - Public
    - accessed by members of the same class, members of other classes in the same package
  - Default
    - accessed by members of the same class, members of other classes in the same package

- Member access rights
  - Public members
    - accessed from everywhere
  - Protected members
    - accessed by members of the same class, members of a subclass, members of other classes in the same package
  - Private members
    - accessed only from the same class
    - are not inherited
  - Default
    - accessed by members of the same class, members of other classes in the same package

- Local variables do not have access modifiers
- Public and protected members retain their access modifiers when they are inherited to other classes

modifiers	**class**	**package**	**subclass in package**	**subclass in different package**	**world**
private	YES	-	-	-	-
default	YES	YES	YES	-	-
protected	YES	YES	YES	YES	-
public	YES	YES	YES	YES	YES

- final modifier
  - Applies to class, method and variable
  - Class: a final class cannot be extended
  - Method: a final method cannot be overridden
  - Variable: a final variable cannot be modified
    - If a final variable is an object, it is the reference that must stay the same not the object

- Does it compile?

```
final class TestFinal {
   public void getStatus() {}
}

final class ExtendedFinal extends TestFinal {
   public static void main(String[] args) {
   }
}
```

- Does it compile?

```
class FinalLearner {
   final void live() {}
   public void speak() {}
   public void learn() {}
}
class TestFinal extends FinalLearner {
   final void live() {}
   public void speak() {}
   final public void learn() {}
}
```

- What are the errors?

```
classAlien{
   final int x = 0;
   final int age;
   final String planet;
   Alien(String star){
        planet= star;
        age= 100;
    }

    void do Something(){
        final int i = 5;
        i = 10;
        x = 5;
    }
}
```

- What are the errors?

```
class Ball(){
    int x=0;
    int y=0;
}

class TestBall{
    final int age= 0;
    final Ball b1 = new Ball();
    final Ball b2 = new Ball();
    void doSomething(){
        b1.x = 10;
        b2.y = 100;
        b1 = new Ball();
    }
}
```

- Conversions
  - A derived class is a superset of the base class
  - A derived class can be converted to the base class but not vice versa
    - Not all the elements in the derived class are present in the base class
    - If converted, information would be lost

s23

Example

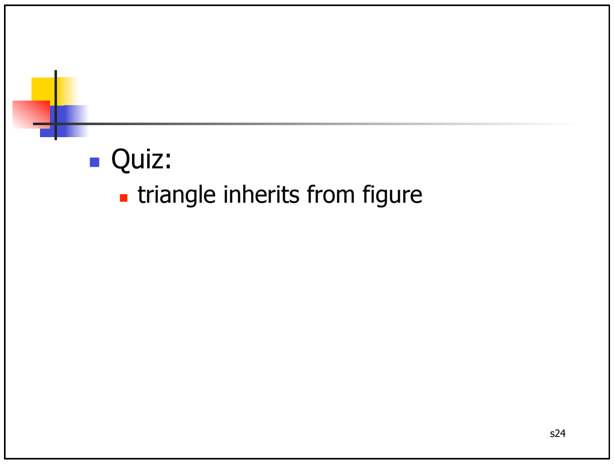
```
public class Figure {
   int area;

   public int getArea() {
      return area;
   }
}
```

```
public class Rectangle extends Figure {
   private int base;
   private int height;

   public void setBase(int x) { base = x; }
   public void setHeight(int x) { height = x; }
   public void computeArea() { area = base * height; }
}
```

```
public class Square extends Figure {
   private int side;

   public void setSide(int x) { side = x; }
   public void computeArea() { area = side * side; }
}
```

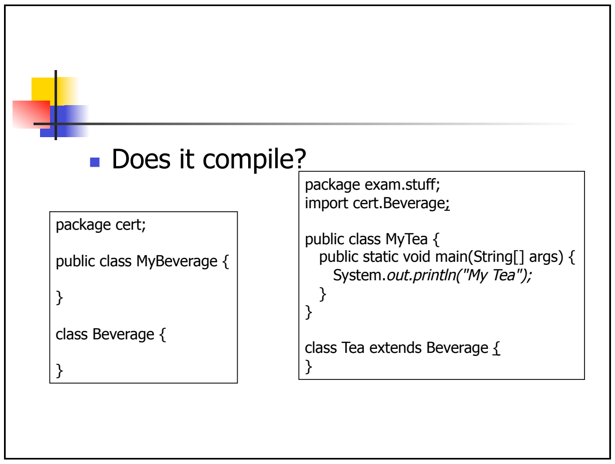
Example

```
public class Main {
   public static void main(String[] args) {
      Square square = new Square();
      square.setSide(5);
      square.computeArea();
      System.out.println("Square area: " +  square.getArea());
      square.side; //not valid

      Rectangle rectangle = new Rectangle();
      rectangle.setBase(10);
      rectangle.setHeight(5);
      rectangle.computeArea();
      System.out.println("Rectangle area: " +  rectangle.getArea());
   }
}
```



- Quiz:
  - triangle inherits from figure

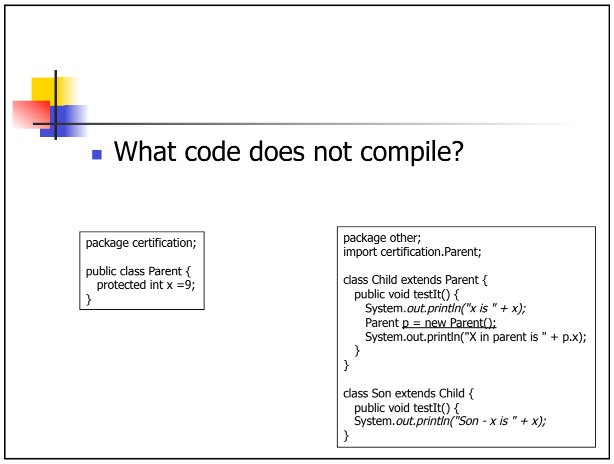
s24



- Does it compile?

```
package cert;

public class MyBeverage {

}

class Beverage {

}
```

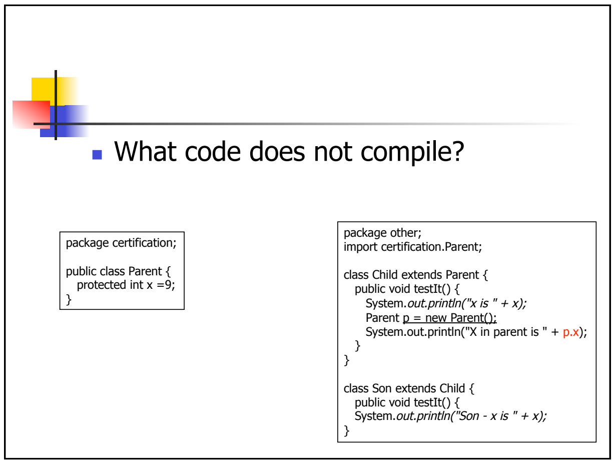
```
package exam.stuff;
import cert.Beverage;

public class MyTea {
   public static void main(String[] args) {
      System.out.println("My Tea");
   }
}

class Tea extends Beverage {
}
```



- What code does not compile?

```
package certification;

public class Parent {
   protected int x =9;
}
```

```
package other;
import certification.Parent;

class Child extends Parent {
   public void testIt() {
      System.out.println("x is " + x);
      Parent p = new Parent();
      System.out.println("X in parent is " + p.x);
   }
}

class Son extends Child {
   public void testIt() {
   System.out.println("Son - x is " + x);
}
```



- What code does not compile?

```
package certification;

public class Parent {
   protected int x =9;
}
```

```
package other;
import certification.Parent;

class Child extends Parent {
   public void testIt() {
      System.out.println("x is " + x);
      Parent p = new Parent();
      System.out.println("X in parent is " + p.x);
   }
}

class Son extends Child {
   public void testIt() {
   System.out.println("Son - x is " + x);
}
```

What code does not compile?

```
package cert;

public class Animal {
   protected void move() {}
    void eat() {}
}
```

```
package cert;

public class Dog extends Animal {
   public void live() {
      move();
      eat();
   }
}
```

```
package certext;
import cert.Animal;

public class Dog extends Animal{
   public void live() {
      move();
      eat();
   }
}
```

What code does not compile?

```
package cert;

public class Animal {
   protected void move() {}
    void eat() {}
}
```

```
package cert;

public class Dog extends Animal {
   public void live() {
      move();
      eat();
   }
}
```

```
package certext;
import cert.Animal;

public class Dog extends Animal{
   public void live() {
      move();
      eat();
   }
}
```

- Invalid declaration

```
private class DogCatcher {
   default String name;

   DogCatcher(float x) {
      public int y;
      private int z;
   }

   Void goSleep(public int x) {}
}
```

- Invalid declaration

```
private class DogCatcher {
   default String name;

   DogCatcher(float x) {
      public int y;
      private int z;
   }

   Void goSleep(public int x) {}
}
```

- A method cannot be overriden to become more private
- The valid overriden method access modifier is
  - Private->default->protected->public

What code compiles?

```
package cert;

public class Animal {
   private void run() {}
   void eat() {}
   protected void move() {}
   public void go() {}
}
```

```
package cert;

public class Dog extends Animal {
   private void run() {}
   void eat() {}
   public void move() {}
   public void go() {}
}
```

```
package certext;
import cert.Animal;

public class Dog extends Animal{
   protected void run() {}
   protected void eat() {}
   void move() {}
   void go() {}
}
```

What code compiles?

```
package cert;

public class Animal {
   private void run() {}
   void eat() {}
   protected void move() {}
   public void go() {}
}
```

```
package cert;

public class Dog extends Animal {
   private void run() {}
   void eat() {}
   public void move() {}
   public void go() {}
}
```

```
package certext;
import cert.Animal;

public class Dog extends Animal{
   protected void run() {}
   protected void eat() {}
   void move() {}
   void go() {}
}
```

Abstract Classes

- Can be applied to classes and methods
- Abstract classes can have both abstract and non-abstract methods
- An abstract method can not be defined in non-abstract classes
- An abstract class must be extended
- The only allowed access modifiers for abstract members are protected and public

- The class must be declared abstract if any of the following conditions is true
  - The class has one or more abstract methods
  - The class inherits abstract class and the class itself does not yet implement all abstract methods
  - The class declares it implements an interface but does not provide implementations for all methods of such an interface

Example

```
public abstract class Vehicle{
    private String type;
    public abstract void goUpHill();
    public String getType() { return type; }
}

public abstract class Car extends Vehicle{
    public abstract void goUpHill();
    public abstract void goDownHill(int speed) ;
    public void doCarThings() { }
}

public class Mini extends Car{ …}
```

- Quiz
  - Create an abstract superclass and concrete subclass for the matrix application

Interfaces

- Can be used to Standardise the way certain services are accessed
- Separate service specification from service implementation
  - Specified with an in interface
  - Implemented with a class
- An interface can be public, protected, private or default

Interfaces

- An interface can only contain:
  - Constants
    - All fields must be initialised and are implicitly public, static and final
  - Abstract methods
    - All methods are implicitly public and abstract

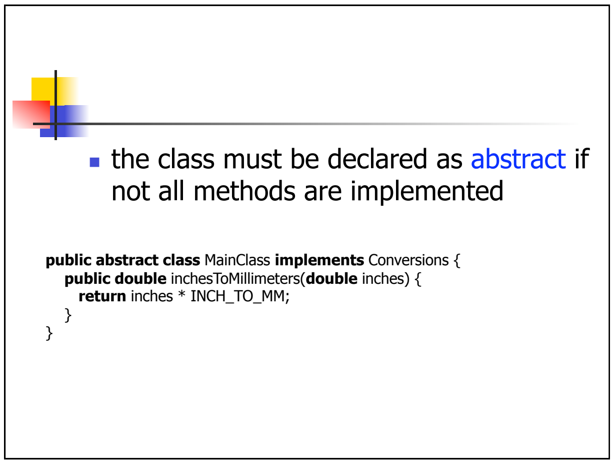
- Does the following code compile?

```
interface A {
   void methodA();
}

public class Temp implements A {
   void methodA() {}
}
```

```
interface Conversions {
  double INCH_TO_MM = 25.4;
  double OUNCE_TO_GRAM = 28.349523125;
  double POUND_TO_GRAM = 453.5924;
  double HP_TO_WATT = 745.7;
  double WATT_TO_HP = 1.0 / HP_TO_WATT;

  public double inchesToMillimeters(double inches);
  public double ouncesToGrams(double ounces);
}
```

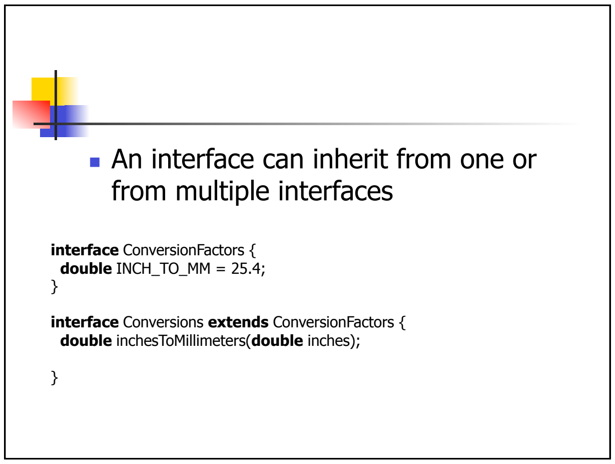
- The keyword implements is used to implement an interface

```
public class MyConversions implements Conversions {
  public double inchesToMillimeters(double inches) {
    return inches * INCH_TO_MM;
  }

  public double ouncesToGrams(double ounces) {
    return ounces * OUNCE_TO_GRAM;
  }

}
```



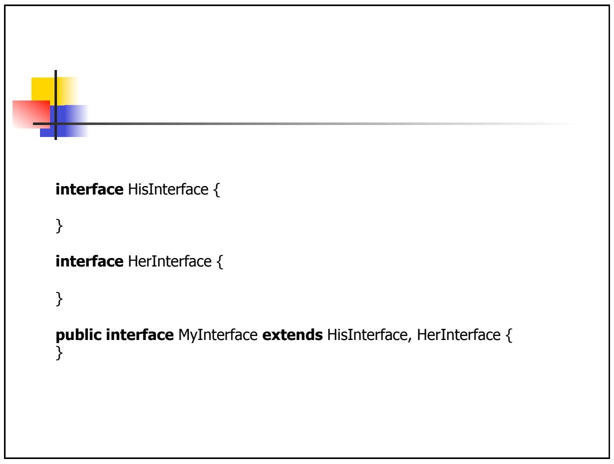
- the class must be declared as abstract if not all methods are implemented

```
public abstract class MainClass implements Conversions {
   public double inchesToMillimeters(double inches) {
      return inches * INCH_TO_MM;
   }
}
```

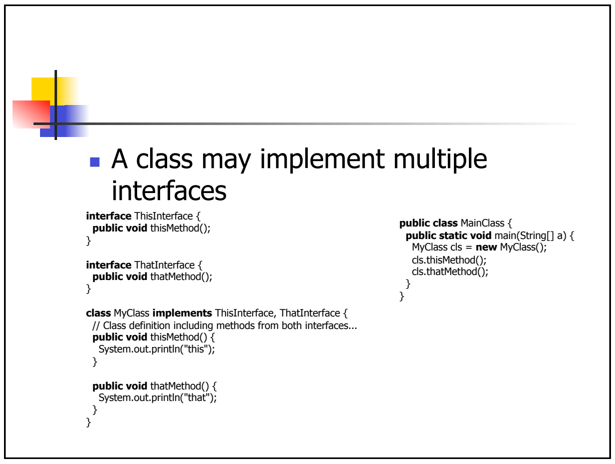


- An interface can inherit from one or from multiple interfaces

```
interface ConversionFactors {
  double INCH_TO_MM = 25.4;
}

interface Conversions extends ConversionFactors {
  double inchesToMillimeters(double inches);

}
```



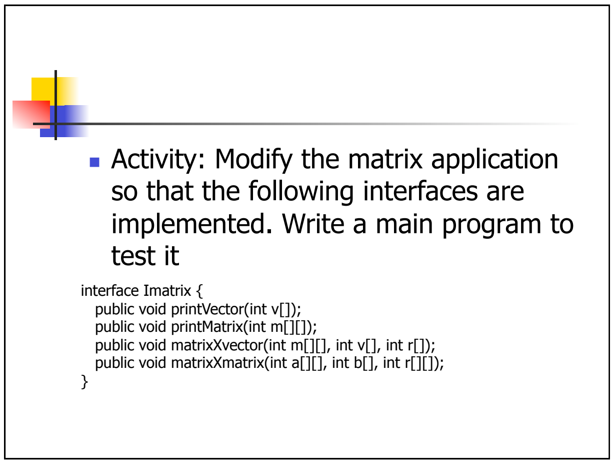
```
interface HisInterface {

}

interface HerInterface {

}

public interface MyInterface extends HisInterface, HerInterface {
}
```



- A class may implement multiple interfaces

```
interface ThisInterface {
  public void thisMethod();
}

interface ThatInterface {
  public void thatMethod();
}

class MyClass implements ThisInterface, ThatInterface {
  // Class definition including methods from both interfaces...
  public void thisMethod() {
    System.out.println("this");
  }

  public void thatMethod() {
    System.out.println("that");
  }
}
```

```
public class MainClass {
  public static void main(String[] a) {
    MyClass cls = new MyClass();
    cls.thisMethod();
    cls.thatMethod();
  }
}
```



- Activity: Modify the matrix application so that the following interfaces are implemented. Write a main program to test it

```
interface Imatrix {
   public void printVector(int v[]);
   public void printMatrix(int m[][]);
   public void matrixXvector(int m[][], int v[], int r[]);
   public void matrixXmatrix(int a[][], int b[], int r[][]);
}
```