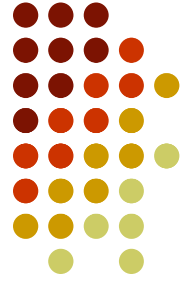
Example using Divide&Conquer Approach: More efficient matrix multiplication



- Two 2x2 (block) matrices can be multiplied
  - C==AB
  - $[c_{11}\ c_{12};\ c_{21}\ c_{22}]==[a_{11}\ a_{12};\ a_{21}\ a_{22}][b_{11}\ b_{12};\ b_{21}\ b_{22}]$

- with 7 multiplications and 18 additions:
  - $Q_1 = (a_{11}+a_{22})(b_{11}+b_{22})$
  - $Q_2 = (a_{21}+a_{22})b_{11}$
  - $Q_3 = a_{11}(b_{12}-b_{22})$
  - $Q_4 = a_{22}(-b_{11}+b_{21})$
  - $Q_5 = (a_{11}+a_{12})b_{22}$
  - $Q_6 = (-a_{11}+a_{21})(b_{11}+b_{12})$
  - $Q_7 = (a_{12}-a_{22})(b_{21}+b_{22})$.

$c_{11} = Q_1+Q_4-Q_5+Q_7$

$c_{21} = Q_2+Q_4$

$c_{12} = Q_3+Q_5$

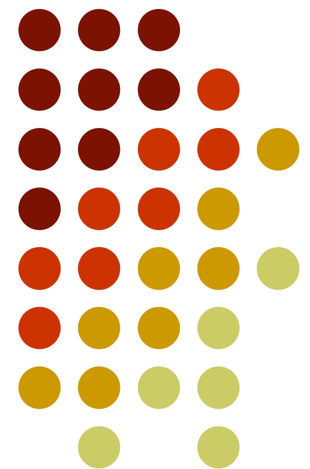
$c_{22} = Q_1+Q_3-Q_2+Q_6$

- (Strassen 1969, Press et al. 1989).
- Which results in the recursive algorithm with time cost $T(n)=7.T(n/2)+O(n^2)$
- Solving recurrence gives: $T(n)=O(n^{\log_2 7})=O(n^{2.81})$.

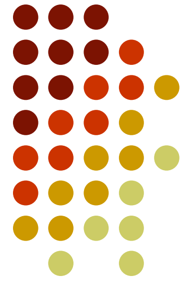
CMPT 225

Inheritance

Reusing the code

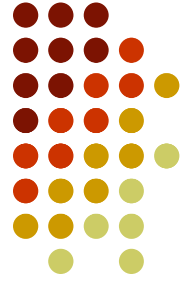


Inheritance



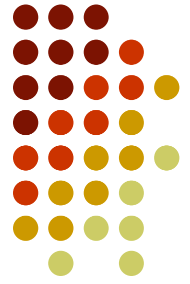
- Inheritance is a relationship between classes whereby one class can derive the behaviour and structure of another class
- Superclass or base class
  - A class from which another class is derived
- Subclass, derived class, or descendant class
  - A class that inherits the members of another class
- Benefits of inheritance
  - to capture hierarchies that exist in a problem domain
  - enables the reuse of existing classes which reduces the effort necessary to add features to an existing object
  - reduces duplicated code
    - cheaper to maintain (changes more localized)
  - to specialize an existing class
    - e.g. subclassing Swing components is encouraged in Java

Inheritance Fundamentals



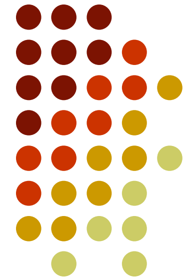
- A subclass inherits **all** members (private and public) from the superclass, but private members cannot be accessed directly!
- Methods of a subclass can call the superclass's public methods (and use public data members)
- Clients of a subclass can invoke the superclass's public methods

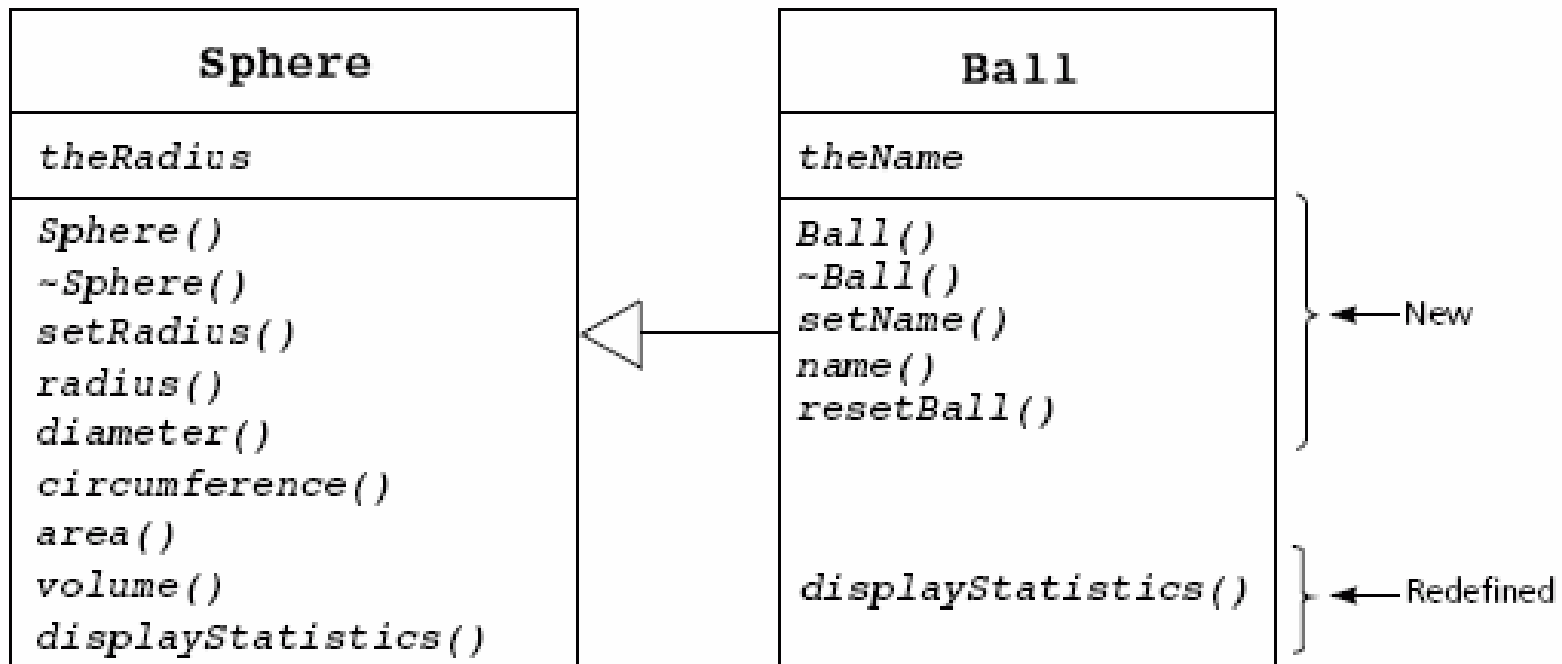
Inheritance Fundamentals



- Additional data members and methods can be defined in subclasses
- Superclass methods can be **overwritten/overridden** in the subclass
  - That is, a subclass may include its own definition of a method that exists in the superclass (they must have same declaration, not only the same name!)
- An overridden method
  - Instances of the subclass will use the new method
  - Instances of the superclass will use the original method

Example





The subclass ***`Ball`*** inherits members of the superclass ***`Sphere`*** and overrides and adds methods

Example

Sphere

constructors
setRadius(newRadius)
radius()
diameter()
circumference()
theRadius
area()
volume()
equals(rhs)
displayStatistics()

Inherits

Overrides

Overrides

Ball

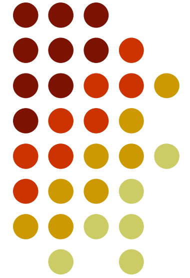
theName
constructors
setName(newName)
name()
resetBall(newRadius,newName)

New

equals(rhs)
displayStatistics()

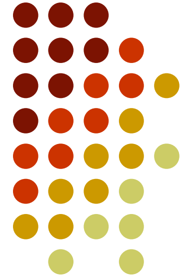
The subclass ***`Ball`*** inherits members of the superclass ***`Sphere`*** and overrides and adds methods

Example (Java)



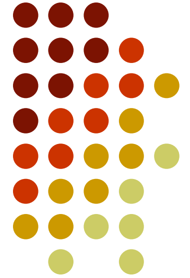
```
public class Sphere {
  private double theRadius;
  public Sphere() {
    setRadius(1.0);
  } // end default constructor

  public Sphere(double initialRadius) {
    setRadius(initialRadius);
  } // end constructor

  public boolean equals(Object rhs) {
    return ((rhs instanceof Sphere) && (theRadius == ((Sphere)rhs).theRadius));
  } // end equals

  public void setRadius(double newRadius) {
    if (newRadius >= 0.0) {
      theRadius = newRadius;
    } // end if
  } // end setRadius

  public double radius() {
    return theRadius;
  } // end radius
```

Example (Java)



```java
  public double diameter() {
    return 2.0 * theRadius;
  } // end diameter

  public double circumference() {
    return Math.PI * diameter();
  } // end circumference

  public double area() {
    return 4.0 * Math.PI * theRadius * theRadius;
  } // end area

  public double volume() {
    return (4.0*Math.PI * Math.pow(theRadius, 3.0)) / 3.0;
  } // end volume

  public void displayStatistics() {
    System.out.println("\nRadius = " + radius() +
                       "\nDiameter = " + diameter() +
                       "\nCircumference = " + circumference() +
                       "\nArea = " + area() +
                       "\nVolume = " + volume());
  } // end displayStatistics
} // end Sphere
```

Example (Java)



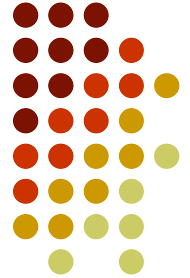
```
public class Ball extends Sphere {
  private String theName;  // the ball's name

  // constructors:
  public Ball() {
  // Creates a ball with radius 1.0 and
  // name "unknown".
    setName("unknown");
  }  // end default constructor

  public Ball(double initialRadius, String initialName) {
  // Creates a ball with radius initialRadius and
  // name initialName.
    super(initialRadius);
    setName(initialName);
  }  // end constructor

  // additional or revised operations:
  public boolean equals(Object rhs) {
    return ((rhs instanceof Ball) &&
            (radius() == ((Ball)rhs).radius()) &&
            (theName.compareTo(((Ball)rhs).theName)==0) );
  }  // end equals
```

specify inheritance

calls default constructor of Sphere first

explicitely calls constructor Sphere(initialRadiud)

Example (Java)

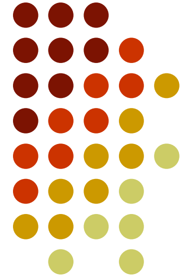


```
  public String name() {
  // Determines the name of a ball.
    return theName;
  }  // end name

  public void setName(String newName) {
  // Sets (alters) the name of an existing ball.
    theName = newName;
  }  // end setName

  public void resetBall(double newRadius, String newName) {
  // Sets (alters) the radius and name of an existing
  // ball to newRadius and newName, respectively.
    setRadius(newRadius);
    setName(newName);
  }  // end resetBall

  public void displayStatistics() {
  // Displays the statistics of a ball.
    System.out.print("\nStatistics for a "+ name());
    super.displayStatistics();
  }  // end displayStatistics

}  // end Ball
```

override method from the superclass

calls displayStatistics() from superclass Sphere

Example C++



```
class Ball : public Sphere {
private:
  string theName;  // the ball's name

public:
  // constructors:
  Ball() {
  // Creates a ball with radius 1.0 and
  // name "unknown".
    setName("unknown");
  }  // end default constructor

  Ball(double initialRadius, string initialName)
    : Sphere(initialRadius) {
  // Creates a ball with radius initialRadius and
  // name initialName.
    setName(initialName);
  }  // end constructor

  // etc.

  virtual void displayStatistics() {
  // Displays the statistics of a ball.
    cout <<"Statistics for a "<<name()<<endl;
    Sphere::displayStatistics();
  }  // end displayStatistics

};  // end Ball
```

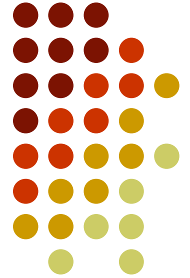
specify inheritance

calls default constructor of Sphere first

explicitely calls constructor Sphere(initialRadiud)

override method from the superclass

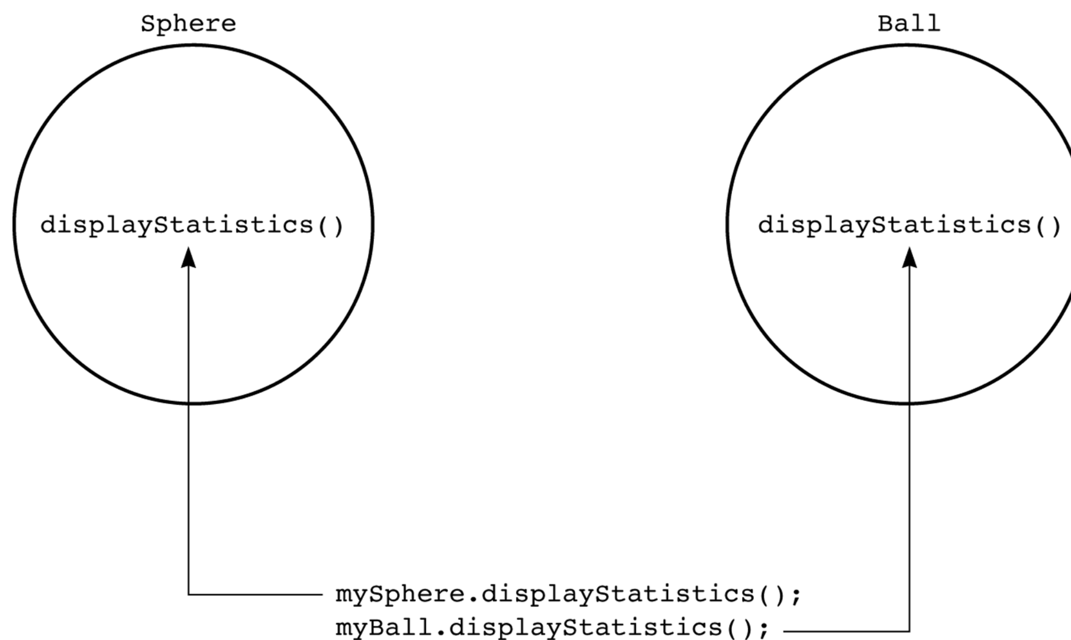
calls displayStatistics() from superclass Sphere



Overriding

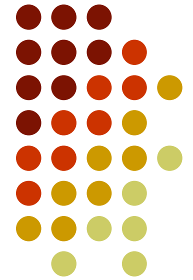
Assume that mySphere is an instance of class Sphere and myBall is an instance of class Ball.

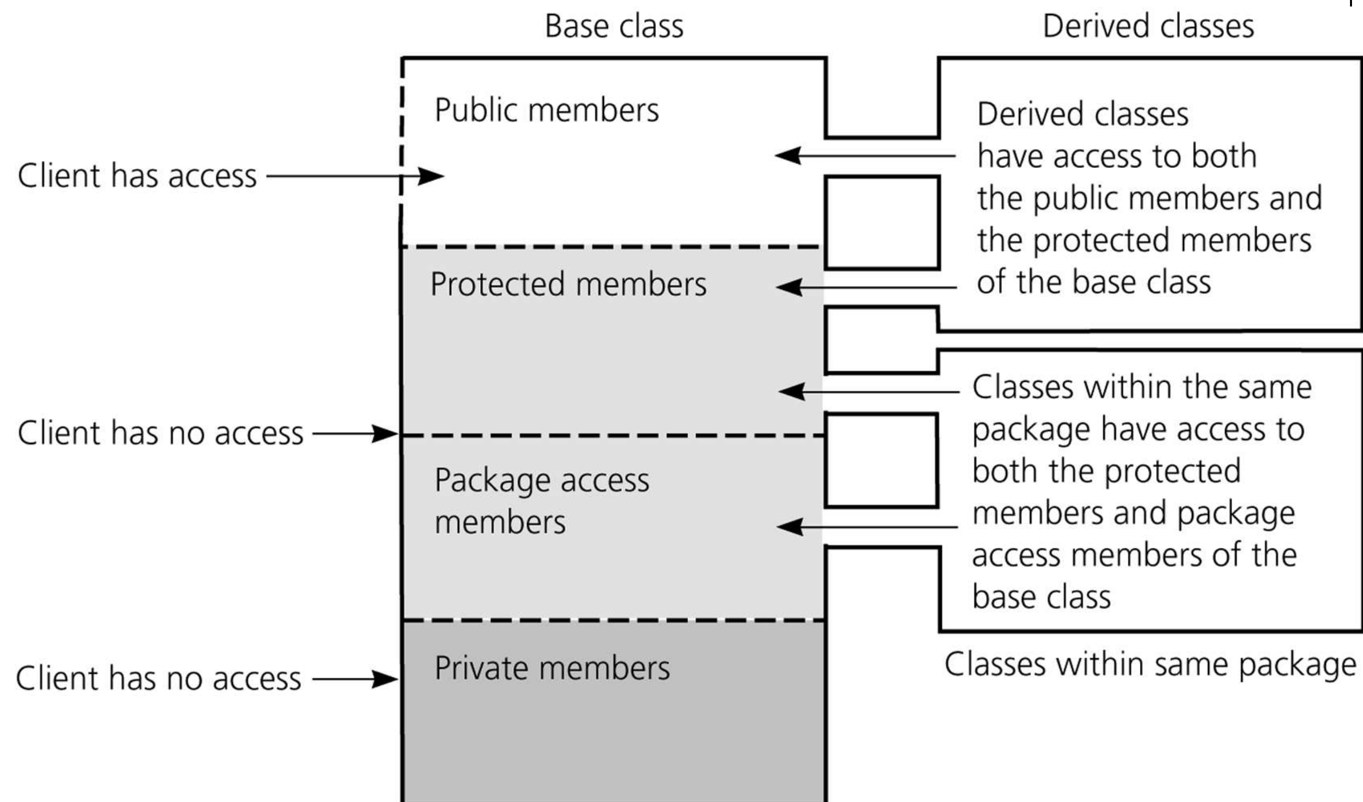
- What happens when we call displayStatistics() on othem?



An object invokes the correct version of a method!

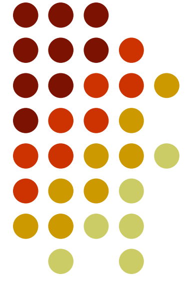
Java Access Modifiers





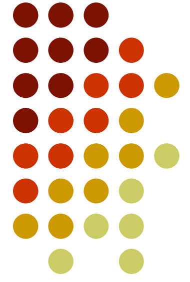
Access to public, protected, package access, and private members of a class by a client and a subclass

Java Access Modifiers (complete list)



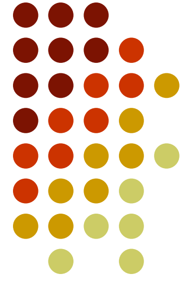
- Membership categories of a class
  - Public members can be used by anyone
  - Members declared without an access modifier (the default) are available to
    - Methods of the class
    - Methods of other classes in the same package
  - Private members can be used only by methods of the class
  - Protected members can be used only by
    - Methods of the class
    - Methods of other classes in the same package
    - Methods of the subclass

C++ Access Modifiers (complete list)



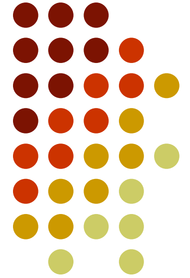
- Note: in C++ sections are used. They are specified by private: public: protected: (instead of prefixing each declaration).
- Membership categories of a class
  - Public members can be used by anyone
  - Private members can be used only by methods of the class
  - Protected members can be used only by
    - Methods of the class
    - Methods of the subclass
- Note: the default section in class in C++ is private:

Is-a and Has-a Relationships

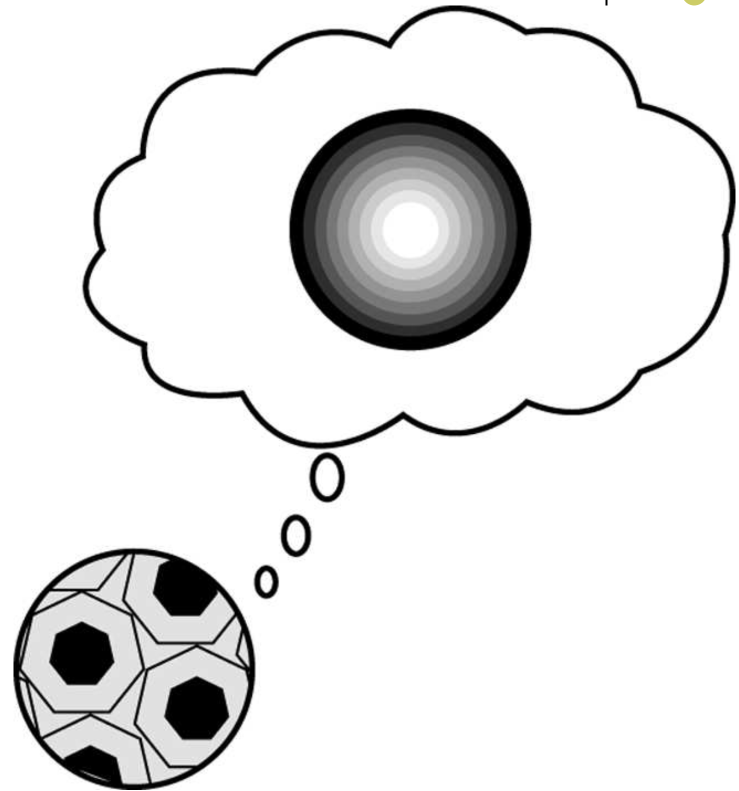


- When designing new classes from existing classes it is important to identify the relationship between the classes
- Two basic kinds of relationships
  - Is-a relationship
    - e.g. a graduate student is a student
  - Has-a relationship
    - e.g. a ball has a color

Is-a Relationship



- Inheritance should imply an is-a relationship between the superclass and the subclass
- Example:
  - If the class `Ball` is derived from the class `Sphere`
    - A ball is a sphere



A ball "is a" sphere