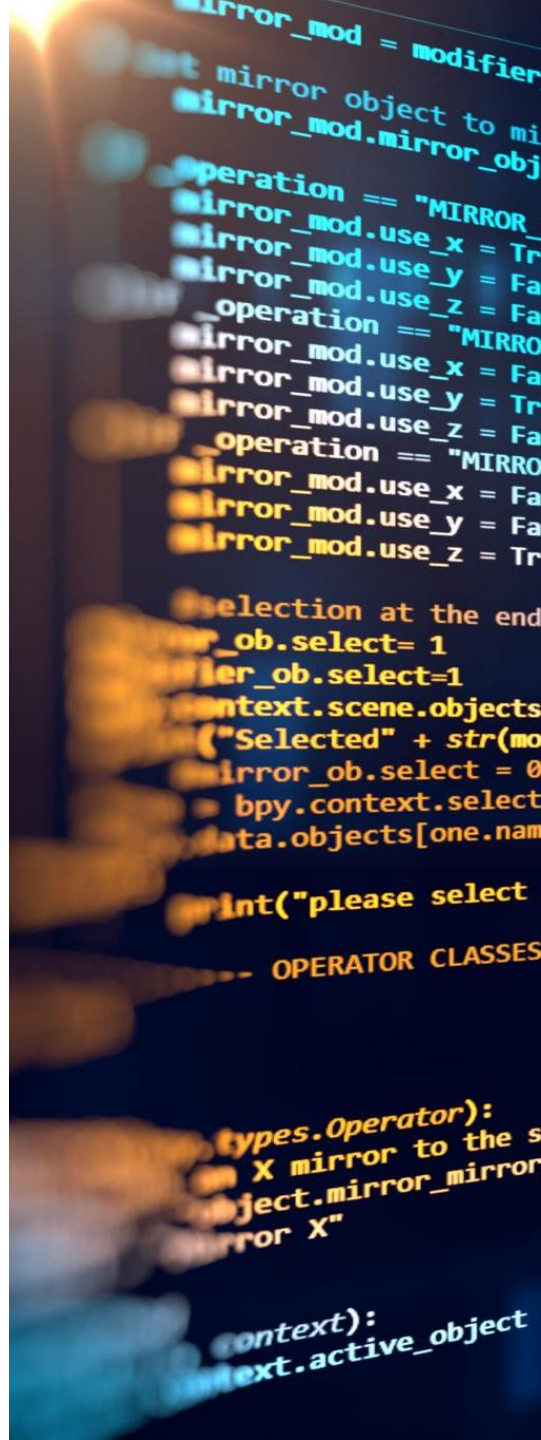
CS 1027
Fundamentals of Computer Science II

Inheritance in Java

Ahmed Ibrahim

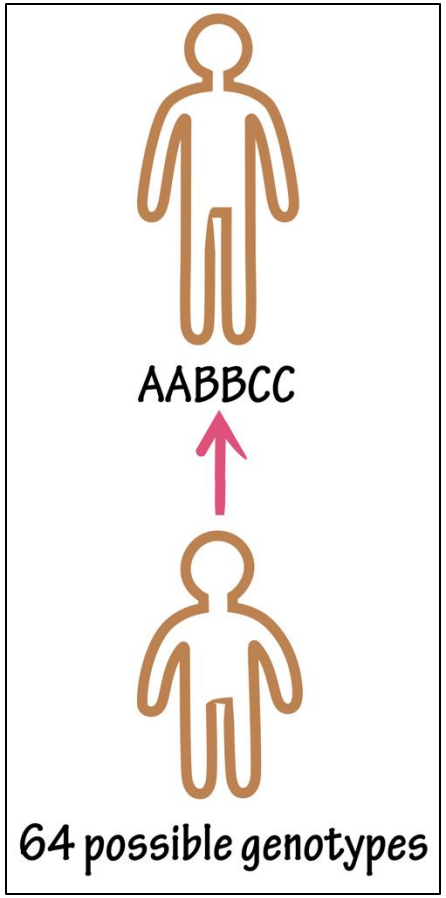


Objectives

- Identify the concept of inheritance
- Create a subclass and override methods from a superclass
- Recognize the importance of the superclass Object and the inheritance hierarchy
- Use the **instanceof** operator to determine the class of an object

Inheritance

- Inheritance is a mechanism for deriving a new class from an existing one.
- Inheritance allows us to reuse existing classes, which is faster and cheaper than writing new classes from scratch



Analogy

Example of Inheritance

- Suppose we have a class called Rectangle that will be used by a program that draws geometric shapes on the screen.
  - Each object of this class stores the **height** and **length** of the rectangle that it represents.
  - These are not just methods, but the backbone of the Rectangle class.
  - They play a critical role in storing and retrieving the **height** and **length** of the rectangle and computing its area.

```
1  public class Rectangle {
2  // Dimensions of the rectangle
3  private int length; private int width;
4
5  public Rectangle(int rLength, int rWidth) {
6    length = rLength;
7    width = rWidth;
8  }
9  public int getLength( ) { return length;
10 }
```

Class Rectangle

Using Inheritance

- Additional methods can be added to enhance the Rectangle class.
- The Rectangle class could serve as a **base class** for other shapes that have a rectangular form.
- Subclasses can inherit properties like length and width from Rectangle, and they also have the freedom to modify or extend the functionality, thereby creating more specialized behaviors.

```
1  public class Rectangle {
2  // Dimensions of the rectangle
3  private int length; private int width;
4
5  public Rectangle(int rLength, int rWidth) {
6    length = rLength;
7    width = rWidth;
8  }
9  public int getLength( ) {
10   return length;
11 }
12
13 public int getWidth( ) {
14   return width;
15 }
16 public int area( ) {
17   return length*width;
18 }
19 /* Represent the object as a String */
20 public String toString( ) {
21 return "Rectangle: " +
22 "Length("+ length +") "+"Width("+ width +")";
23  }
24 }
```

String concatenation

Class Rectangle

Derived Class Square

- We want to write a class that represents squares.
- **Squares** are **special rectangles** for which the length and width are the same. Hence, we want a square to have some of the methods of the class Rectangle like the method to compute the area.
- Furthermore, we need to include specific attributes and methods for squares, such as a method to determine the side of a square. This justifies the creation of a separate class for squares. Why???

Example of Inheritance

- The Square class extends the Rectangle class, inheriting its properties and methods.
- It overrides or adds specific behaviors where necessary, such as defining **getSide**() to return the side length.
- This demonstrates how a subclass can reuse and adapt the functionality of a superclass.

```
1  public class Square extends Rectangle {
2  // Length of the diagonal private double diagonal;
3
4  public int getSide( ) {
5    return getWidth( );
6   }
7  public String toString( ) {
8  return "Square: Side(" + getSide( ) + ")";
9   }
10 }
```

Class Square

Inheritance Terminology

- A new class created with the Java keyword extends is called a subclass, child class, or derived class.
- A subclass inherits the attributes and methods of the superclass (also called the parent class or base class).
- A subclass can add new attributes or methods, i.e., it can **extend** the parent class.

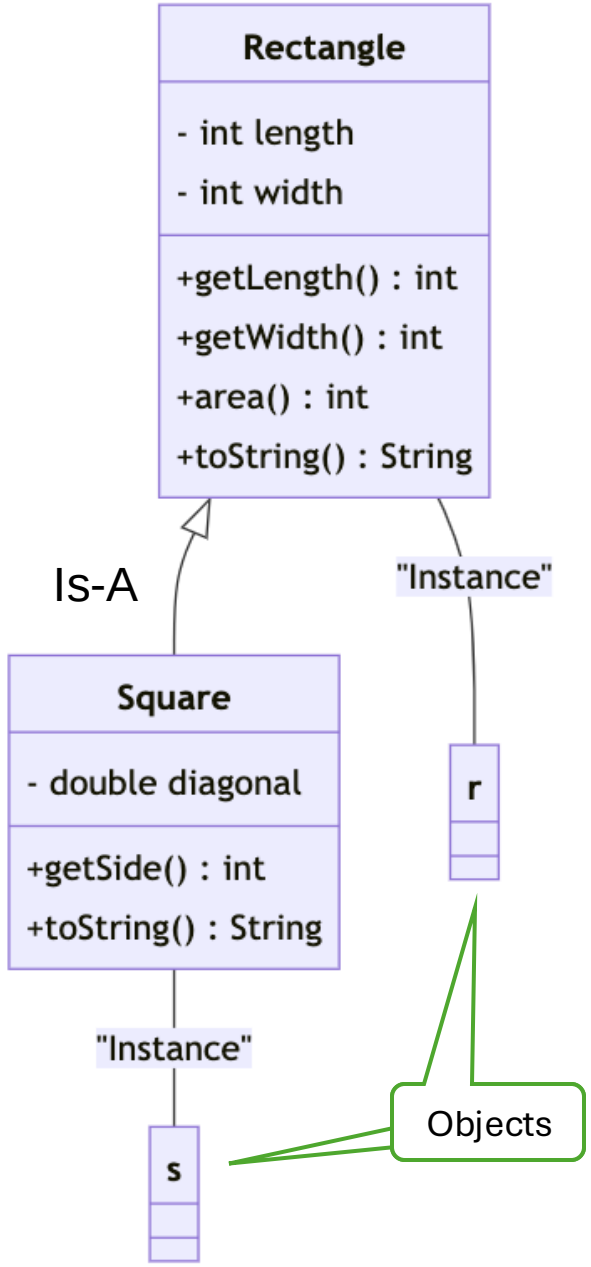
Inheritance

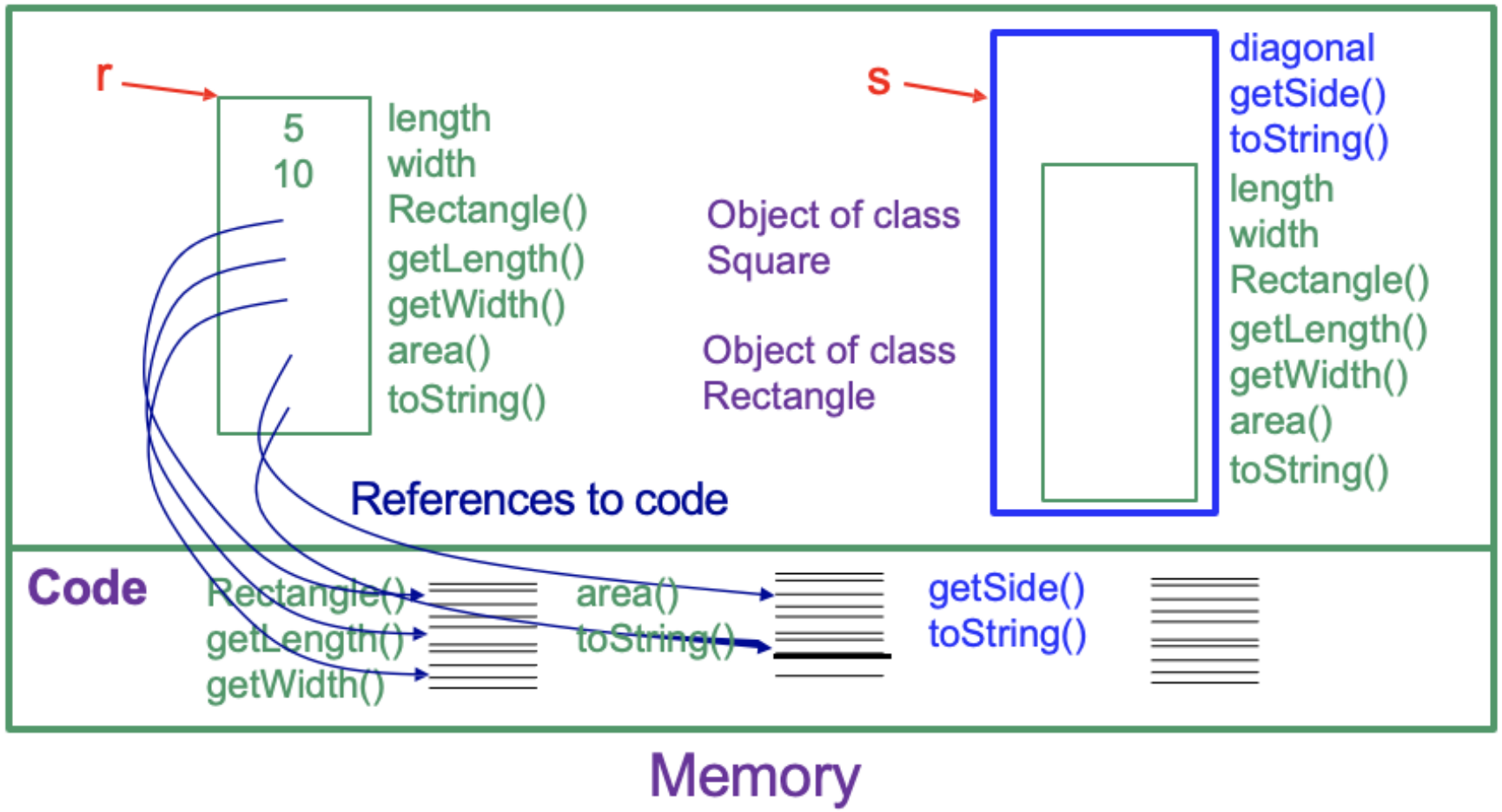
- The following Java fragment will create the two objects shown.

```
Rectangle r = new Rectangle (5,10);
Square s = new Square(4);
```

Memory Table

Object	Class	Attributes	Methods
r	Rectangle	length, width	getLength(), getWidth(), area(), toString()
s	Square	length, width, diagonal	getLength(), getWidth(), area(), getSide(), toString()




r
5
10
length
width
Rectangle()
getLength()
getWidth()
area()
toString()
Object of class
Square
Object of class
Rectangle
s
diagonal
getSide()
toString()
length
width
Rectangle()
getLength()
getWidth()
area()
toString()
References to code
Code
Rectangle()
getLength()
getWidth()
area()
toString()
getSide()
toString()
Memory

Square Constructor

- The Square constructor initializes the square's side and calculates the diagonal.
- It calls the constructor of the Rectangle superclass using super(side, side) to set the length and width properties.
- The diagonal is then computed using the **formula side * 1.4142** (an approximation of $\sqrt{2}$).

```
1  public class Square extends Rectangle {
2  // Length of the diagonal private double diagonal;
3
4  public Square(int side) {
5  // Calls the constructor of the superclass super(side, side);
6   diagonal = (double) side * 1.4142;
7  }
8  public int getSide( ) {
9    return getWidth( );
10  }
11 public String toString( ) {
12 return "Square: Side(" + getSide( ) + ")";
13  }
14 }
```

Square Class

The super Reference

- super is a reserved word used in a **derived class** to refer to its **parent class**.

  It allows us to access instance variables and/or methods of the parent class.
- Invoking the parent's constructor: the first line of a child's constructor should be
  - super(...);
- Invoking other parent methods:

  super.methodName(...);

```java
// Superclass: Rectangle
public class Rectangle {
    private int length;
    private int width;

    // Constructor for Rectangle
    public Rectangle(int length, int width) {
        this.length = length;
        this.width = width;
    }

    // Method to calculate area
    public int area() {
        return length * width;
    }

    // Method to display dimensions
    public void displayDimensions() {
    System.out.println("Length: " + length + ", Width: " + width);
    }
}
```

Rectangle Class

```java
// Subclass: Square
public class Square extends Rectangle {
    private double diagonal;

    // Constructor for Square
    public Square(int side) {
        // Calling the parent class (Rectangle) constructor
        super(side, side);
        // Calculating the diagonal
        this.diagonal = side * Math.sqrt(2);

    // Method to display additional square information
    @Override
    public void displayDimensions() {
      // Calling the parent class method
        super.displayDimensions();
        System.out.println("Diagonal: " + diagonal);
    }
}
```

Square Class

Using this Keyword

- this is a reserved word used within an **instance method or a constructor** to refer to the **current object**. It is often used to access instance variables and methods of the same class, especially when there is a **naming conflict**.
- **Using this to access instance variables –** When a method or constructor has parameters with the same name as instance variables, this helps differentiate them.
  - Example: this.variableName = variableName;
- **Invoking another constructor in the same class –** Use this(…) as the first line of the constructor to call another constructor within the same class.
  - Example: this(parameter1, parameter2);
- **Accessing methods of the current object**:
  - Example: this.methodName(…);

Casting Primitive Types

- We can use casting to convert some primitive types to other primitive types
- Example:
  - int side;
  - double diagonal;
  - diagonal = (double) side * 1.4142;

Casting

- Casting for primitive types changes the binary representation of a value. For the above example, casting changes an int (4 bytes) into a double (8 bytes).

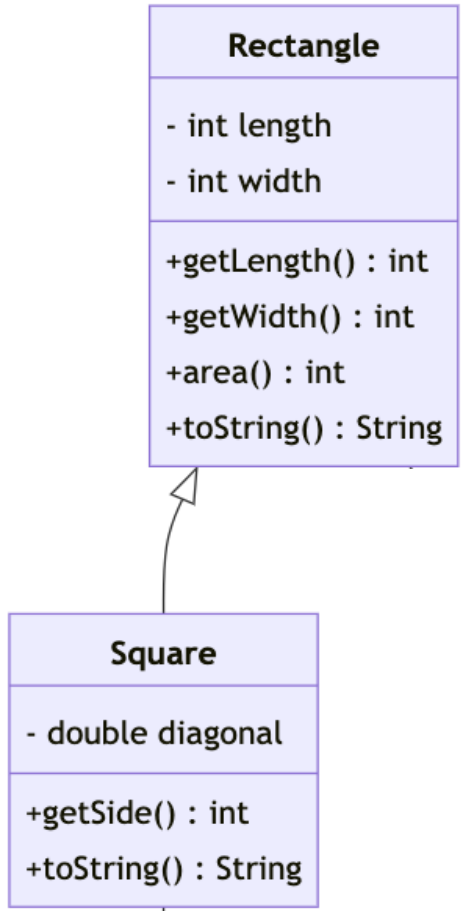
Inheriting Visibility

- public variables and methods: Children's classes can access them directly
- private variables and methods: Children's classes cannot access them directly
- protected variables may be accessed directly by any class in the same package or subclass. Hence, children's classes can access protected variables and methods of a parent class.
- A Java package is a group of classes, all of which start with the statement
  - package package_name;
    - where package_name is the name of the package.

Is-a Relationship

- A derived class is a more specific version of the original class.
- So, if A is a subclass of B, then an object of type A is also an instance of class B.
- Example: An object of class Square is also an object of class Rectangle
  - Is it true that a Rectangle object is also a Square?

Reverse Is Not True



Discussion

Why extend an existing class, i.e., why not just change the existing class by adding the new attributes and methods?

Why extend an existing class?

- **Maintainability**: Inheritance keeps the original class unchanged, making it easier to manage and less prone to errors.
- **Reusability**: It allows using existing functionality and adding new features, reducing code duplication.
- **Flexibility**: Subclasses can adapt or extend behavior without modifying the original class.
- **Encapsulation**: It hides the superclass's implementation, exposing only relevant methods to subclasses.

Discussion

Can you think of more examples of classes we can model with an inheritance relationship?

More Examples

Vehicle Hierarchy:

- Vehicle (Superclass): Common properties like **speed**, **fuel**, **capacity**, and methods like **start**(), **stop**().
- Car (Subclass): Inherits Vehicle and adds attributes like **numberOfDoors**, **trunkSize**.
- Bicycle (Subclass): Inherits Vehicle and adds methods like **pedal**(), **ringBell**().

Example: BankAccount Class

- Suppose we have a class BankAccount with attributes

  private String accountNumber; private double balance;

  and public methods

  Deposit(), withdraw(), printBalance(), getBalance(), toString()
- What attributes and methods of the BankAccount class can be accessed *directly* by code in its subclasses?
- Suppose that we want to derive two new bank account classes: SavingsAccount and CheckingAccount. What new attributes might we have in the subclasses SavingsAccount and CheckingAccount?
  - Examples:
    - in SavingsAccount : interestRate
    - in CheckingAccount : transactionCount

Example: BankAccount constructor:

```
public BankAccount(double initialAmount, String accountNumber) {
    this.balance = initialAmount;
    this.accountNumber = accountNumber;
}
```

CheckingAccount constructor:

```
public CheckingAccount(double initialAmount, String accountNumber) {
super(initialAmount, accountNumber);
transactionCount = 0;
}
```

Example: BankAccount Class

- What new methods might we then have in subclasses SavingsAccount and CheckingAccount?
  - In SavingsAccount:
    - addInterest
    - getInterestRate
  - In CheckingAccount:
    - deductFees
    - deposit
    - withdraw

Overriding Methods

- A derived class can define a method with the same signature (same name and number and types of parameters) as a method in the parent class
  - The child's method overrides the parent's method
  - Example: methods deposit and withdraw in CheckingAccount override deposit and withdraw of BankAccount
  - Example: method toString in Square overrides toString of Rectangle

Overriding Methods

- Which method is actually executed at run time?
  - It depends on which object is used to invoke the method
  - Example:
    - Rectangle r = new Rectangle(4,5); Square s = new Square(5);
      System.out.println(r.toString( ));
    - System.out.println(s.toString( ));
- Which toString method is invoked in the last two statements?
  - **r.toString()** will invoke the toString method of the Rectangle class because r is an instance of Rectangle.
  - **s.toString()** will invoke the toString method of the Square class because s is an instance of Square.

Review the super Reference

- super allows us to invoke a method of the parent class that was overridden in the child class
  - Example:

```
public void deposit (double amount) {
balance = balance + amount; }
public void deposit (double amount) {
transactionCount++;
super.deposit (amount); }
```

- What would happen if we did not have the super reference here?
  - If we did not have the super reference here, the overridden deposit method in the child class would not call the deposit method from the parent class.
  - Instead, only the code inside the child class's deposit method would be executed.

Superclass Variables

- Are these statements valid?
  - Square s = new Square(5); Rectangle r = s;
  - Rectangle r1 = new Square(6);
  - Square s1 = new Rectangle(2,3);
- Recall that class Square extends class Rectangle, so all instance variables and methods of class Rectangle are part of an object of the class Square (see next slide).

Example: BankAccount constructor:

```
public BankAccount(double initialAmount, String accountNumber) {
    this.balance = initialAmount;
    this.accountNumber = accountNumber;
}
```

CheckingAccount constructor:

```
public CheckingAccount(double initialAmount, String accountNumber) {
super(initialAmount, accountNumber);
transactionCount = 0;
}
```

Square Inherits Rectangle

- The Square class inherits from the Rectangle class using the extends keyword.
- This means Square gains access to all the methods and attributes of Rectangle while adding its own specific attributes and methods, such as the **diagonal** and **getSide**().
- The **super(side, side)** call in the Square constructor shows how the Square class utilizes the Rectangle constructor to initialize its dimensions. This demonstrates the 'is-a' relationship, where a Square is a specialized Rectangle form.

```
public class Square extends Rectangle {
   // Length of the diagonal
   private double diagonal;
   public Square(int side) {
        // Calls the constructor of the superclass
        super(side, side);
       diagonal = (double) side * 1.4142;
   }
   public int getSide( ) {
        return getWidth( );
   }
   public String toString( ) {
       return "Square: Side(" + getSide( ) + ")";
   }
}
```

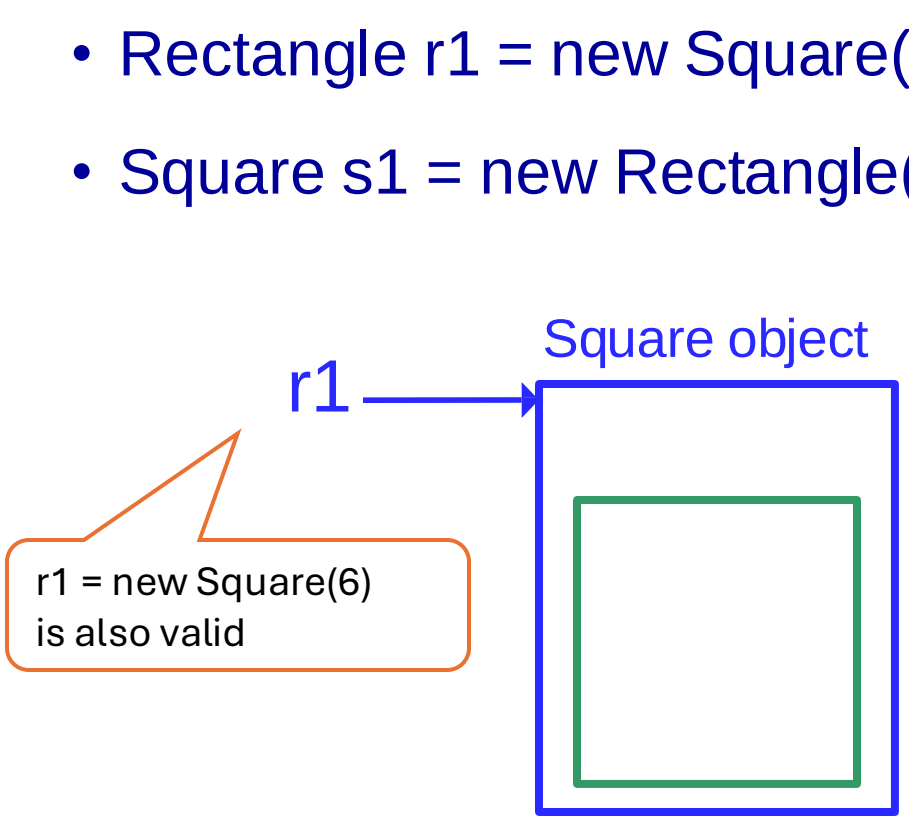
```
public class Rectangle {
   // Dimensions of the rectangle
      private int length;
      private int width;
      public Rectangle(int rLength, int rWidth) {
                length = rLength;
                width = rWidth;
      }
      public int getLength( ) {
                return length;
      }
     public int getWidth( ) {
                return width;
      }
      public int area( ) {
      return length*width;
      }
     /* Represent the object as a String */
      public String toString( ) {
      return "Rectangle: " +
                "Length("+ length +") "+"Width("+
                width +")";
      }
}
```

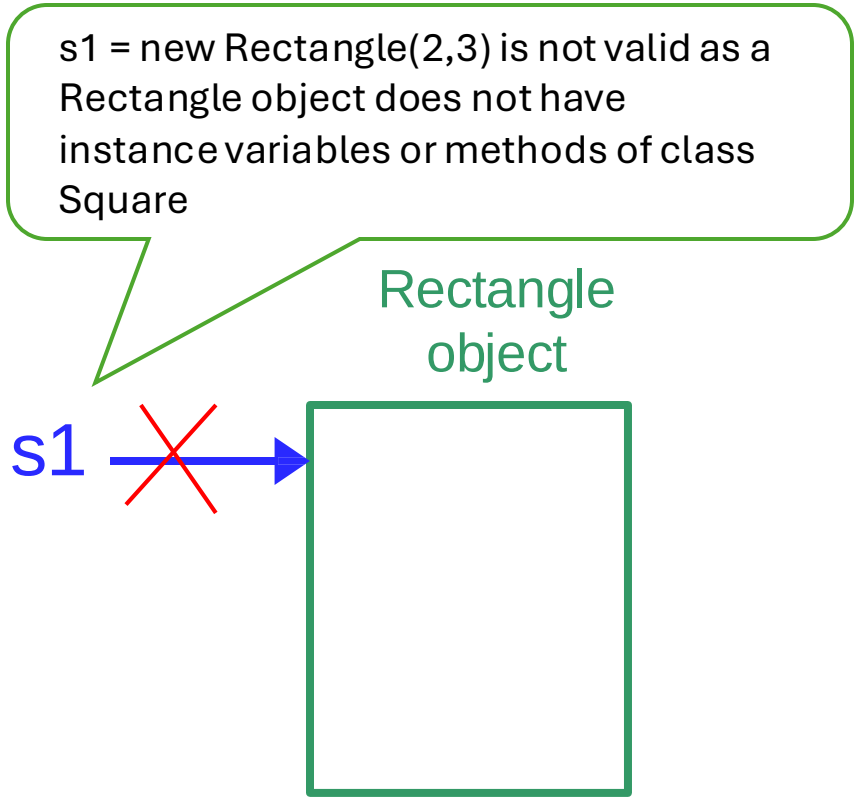
Superclass Variables

- A variable of the superclass type may reference an object of a subclass type.
- However, a variable of the subclass type cannot reference an object of the superclass type.

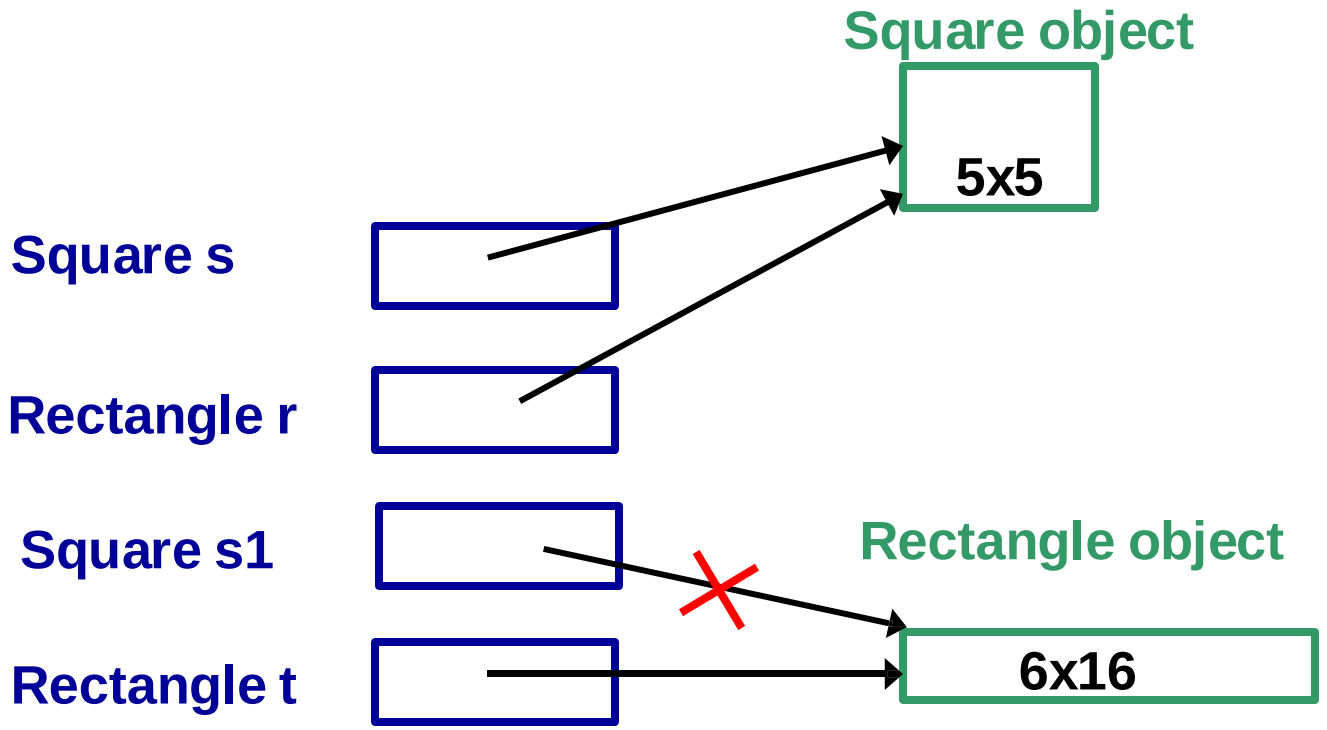
Superclass Variables

- Rectangle r1 = new Square(6);
- Square s1 = new Rectangle(2,3);





Superclass Variables



Superclass Variables

- The type of an object is determined when it is created and **does not change** throughout the execution of a program.