**Common Features in Classes**

- Certain types of objects have things in common
  - Cars/trucks/motorcycles
  - Savings/checking/investment accounts

- In Java, such similarities are exploited by **inheritance**
  - Code can be made simpler and more useful
  - Similarities are written into the **super-class** (parent)
  - Each **sub-class** (child) inherits all the similarities, and then can have particular differences of its own
    - Child will have **methods and variables** from the parent

---

**Inheritance in Pictures and in Code**

- Inheritance can be represented as a diagram, with arrows from children to parents
  - Each child is a **more specific** kind of parent object
  - The code for a child class uses the Java **extends** keyword to achieve this

```java
class Vehicle {
    // usual data and methods
}

class Car extends Vehicle {
    // more data and methods
}

class Motorcycle extends Vehicle {
    // still more data and methods
}
```

---

**Hierarchies of Inheritance**

- Child objects can **themselves** have children
  - For example, a ServiceBusiness extends the general Business class of object
  - HomeBasedService extends ServiceBusiness

```
Business is an ancestor of all classes below it

HomeBasedService is a descendant of all classes above it

Classes on separate branches have NO ancestor/descendant relationship to each other
```
Constructors and the `super` Command

- All methods and variables of a parent are inherited, except for the constructor method.
- Main reason: constructors must have same name as class itself.
- The child runs its own constructor, and if it wants to use the parent constructor it uses the `super` reference.
- Can be used to refer to methods/variables from parent class:
  - `super()` is the parent constructor.
  - `super.methodName()` runs a method from the parent explicitly (here, `super` might be optional).

Running the `super` Constructor

```java
class Account {
    private double balance;

    Account(double openingBalance) {
        balance = openingBalance;
    }
}
```

```java
class MoneyMarketAccount extends Account {
    private double annualInterestRate;

    MoneyMarketAccount(double openingBalance, double rate) {
        super(openingBalance);
        annualInterestRate = rate;
    }
}
```

Scope and Inheritance

```java
class Account {
    private double balance;
    Account(double openingBalance) {
        balance = openingBalance;
    }
}
```

```java
class MoneyMarketAccount extends Account {
    private double annualInterestRate;
    MoneyMarketAccount(double openingBalance, double rate) {
        super(openingBalance);
        annualInterestRate = rate;
    }
    System.out.println("Balance is: " + balance);  // ERROR!
}
```

The protected Scope

```java
class Account {
    protected double balance;
    Account(double openingBalance) {
        balance = openingBalance;
    }
}
```

```java
class MoneyMarketAccount extends Account {
    protected double annualInterestRate;
    MoneyMarketAccount(double openingBalance, double rate) {
        super(openingBalance);
        annualInterestRate = rate;
    }
    System.out.println("Balance is: " + balance);  // ERROR!
}
```
Overriding Methods

- If we want to replace a method in a parent class we can create a new one to override it.
- Must have same method signature: exact same access, name and list of parameters for input or output.
- For example if the parent contains a method:

  ```java
  public void doX(int y) {} // example method
  ```

  The child can override it by declaring own method with:
  1. Same name: `doX`
  2. Same parameter type: `int`
  3. Same output type: `void`
  4. Same access: `public`

An Example of Using Inheritance

```java
import java.awt.Color;

public class Oval extends JComponent {
    public Oval( int x, int y, int w, int h ) {
        super();
        setBackground(Color.black);
    }

    public void paint( Graphics g ) {
        g.setColor(getBackground());
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
        paintChildren(g);
    }
}
```

```java
import java.awt.Color;

public class RedDot extends Oval {
    public RedDot( int x, int y, int d ) {
        super(x, y, d, d);
        setBackground(Color.red);
    }

    public void flatten() {
        setSize(getWidth() + 10, getHeight() - 10);
        repaint();
    }
}
```

Containers and JComponents

Java Container class is a "box" for graphical objects.
- We have used many of its methods, without knowing it.
- Has a sub-class, JComponent, which extends it to produce the on-screen graphics itself.

```java
import java.awt.*;
import javax.swing.JComponent;

public class Oval extends JComponent {
    public Oval( int x, int y, int w, int h ) {
        super();
        setBackground(Color.black);
    }

    public void paint( Graphics g ) {
        g.setColor(getBackground());
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
        paintChildren(g);
    }
}
```

Extending JComponent

- JComponent can itself be extended.
- The paint() method creates graphics on-screen.
- We override paint() to produce our own effects.

```java
import java.awt.*;
import java.awt.Container;

public class Oval extends JComponent {
    public Oval( int x, int y, int w, int h ) {
        super();
        setBackground(Color.black);
    }

    public void paint( Graphics g ) {
        g.setColor(getBackground());
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
        paintChildren(g);
    }
}
```
This Week & Next

- **Meetings this week:**
  - Monday: regular classroom
  - Wednesday: in the CS Lab (16 Wing)

- **Reading 08:** Ch. 9 due Tuesday, Nov 26 at 12:00 PM (Noon)
  - Only participation exercises this time

- **Program 08:** due Monday December 9 at 11:59 PM

- **Office Hours:** Wing 212
  - Monday: 2:15 PM–3:15 PM
  - Wednesday: 12:05 PM–1:00 PM

- **Tuesday is Thursday schedule so this class doesn’t meet**
- **Wednesday is Friday schedule: we meet in lab**
- **Happy Thanksgiving! Safe travels.**