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

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

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

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

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

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

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

Scope and Inheritance

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

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

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

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

- We can fix this by:
  - Creating public methods for access as usual
  - Using protected scope if needed with inheritance
  - protected shares access with all descendant sub-classes (otherwise, acts like default access for most purposes)
Overriding Methods

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

```
public void doX( int y ) { ... }
```

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

```
public void doX( int y ) { ... }
```

### An Example of Using Inheritance

```java
import java.awt.Color;

public class RedDot extends Oval {
    public RedDot(int x, int y, int d) {
        super(x, y, d); // super is called
        setBackground(Color.red); // same method with different implementation
    }

    public void flatten() {
        setSize(getWidth() + 10, getHeight() - 10); // same method with different implementation
        repaint();
    }
}
```

```java
// assume a Window named window exists
RedDot dotty = new RedDot(10, 20, 5);
window.add( dotty );
dotty.setLocation(20, 30);
dotty.flatten();
```

### Containers and JComponents

- `java.awt.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.

```java
import java.awt.
import javax.swing.

public class Oval extends JComponent {
    public Oval(int x, int y, int w, int h) {
        super(); // super is called
        setBackground(Color.black); // same method with different implementation
    }

    public void paint(Graphics g) {
        g.setColor(getBackground());
        g.fillOval(x, y, w, h); // same method with different implementation
        g.clearRect(x, y, w, h);
    }
}
```

### 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 javax.swing.

public class Oval extends JComponent {
    public Oval(int x, int y, int w, int h) {
        super(); // super is called
        setBackground(Color.black); // same method with different implementation
    }

    public void paint(Graphics g) {
        g.setColor(getBackground());
        g.fillRect(x, y, w, h); // same method with different implementation
        paintChildren(g);
    }
}
```
This Week & Next

› **Meetings this week:**
  › Tuesday: Recorded lecture
  › Wednesday: Recorded lecture
  › Friday: Final exam review

› **Reading 08:** Ch. 10 May 08 at 5:00 PM

› **Office Hours (this week):** via the interwebs
  › Monday/Tuesday/Wednesday/Friday: 9:00 AM–11:00 AM
  › [https://kube-0.cs.uwlax.edu:8443/ZombieApocalypseOfficeHours](https://kube-0.cs.uwlax.edu:8443/ZombieApocalypseOfficeHours)

› **Office Hours (finals week):** via the interwebs
  › Monday/Tuesday: 9:00 AM–11:00 AM
  › [https://kube-0.cs.uwlax.edu:8443/ZombieApocalypseOfficeHours](https://kube-0.cs.uwlax.edu:8443/ZombieApocalypseOfficeHours)