Inheritance Tips

When to use what,
when designing or using inheritance.

Designing

Classes, abstract classes and interfaces

Just a class, no inheritance

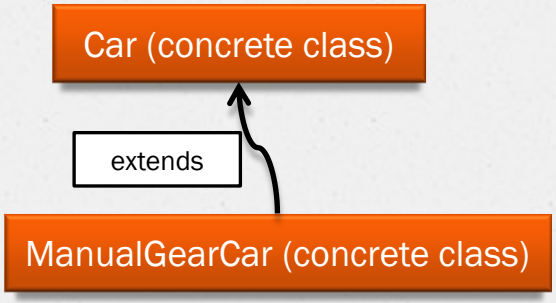
Car (concrete class)

Just a class

- Write a class when it reflects an object in your solution.

```
public class Car {
        private String licenseNumber;
        private String color;
        public String getLicenseNumber() {
                return licenseNumber;
        }
        public String getColor() {
                return color;
        }
}
```

Simple inheritance

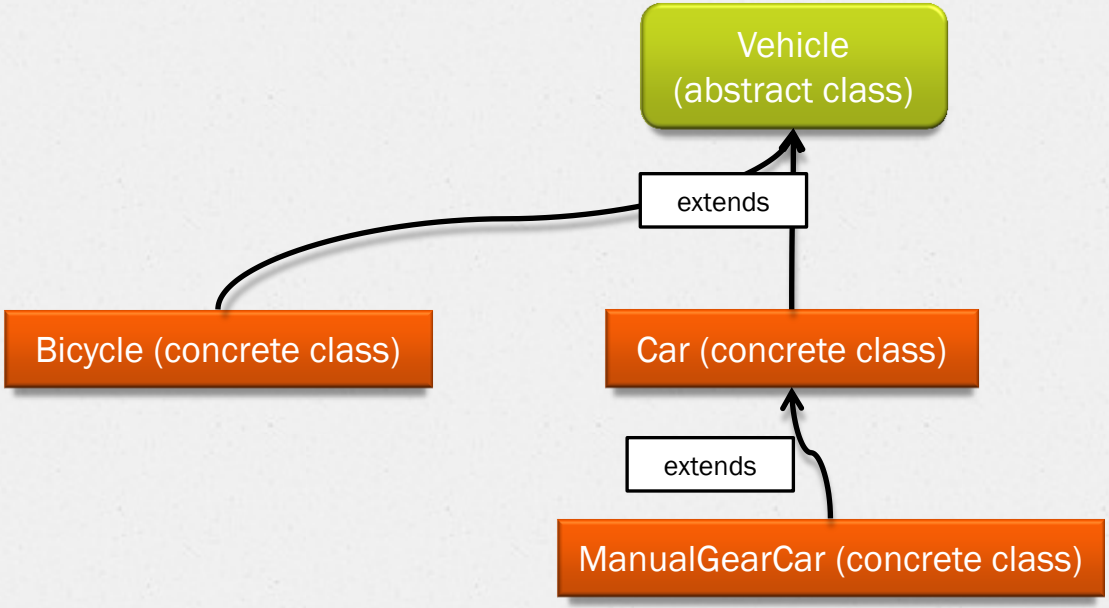


Simple inheritance

- Extend a class when you want to override or extend functionality.

```
public class ManualGearCar extends Car {
        private int gearState;
        public void changeGear(int newGear) {
                gearState = newGear;
        }
}
```

Abstract classes



Abstract classes

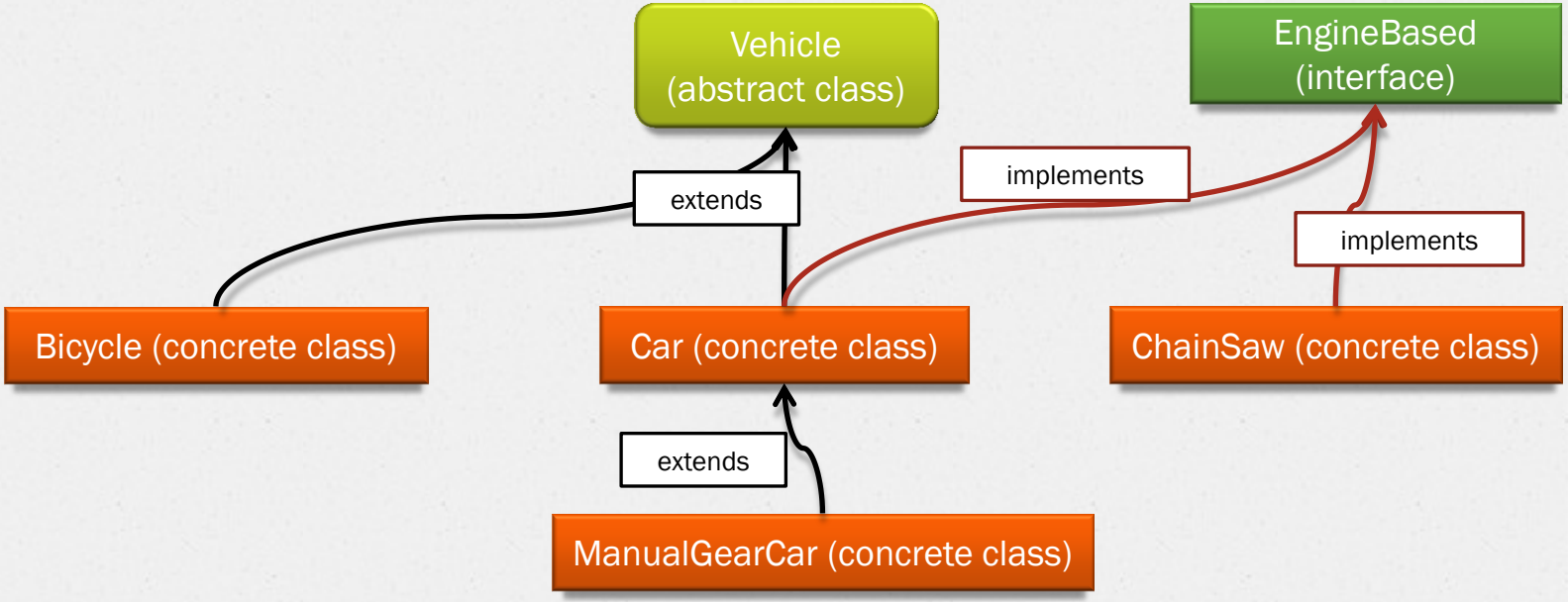
- Write an abstract class if you don't want it to be instantiated.

```
public abstract class Vehicle {
            private String color;
            public String getColor() {
                        return color;
            }
            public abstract void stop();
}
public class Car extends Vehicle {
            private String licenseNumber;
            public String getLicenseNumber() {
                        return licenseNumber;
            }
            public void stop() {
                        hitBreaksPedal();
            }
}
```

Abstract classes (cont'd)

```
public abstract class Vehicle {
        private String color;
        public String getColor() {
                return color;
        }
        public abstract void stop();
}
public class Bicycles extends Vehicle {
        public void stop() {
                pullBothBreakHandles();
        }
}
```

Interfaces



Interfaces

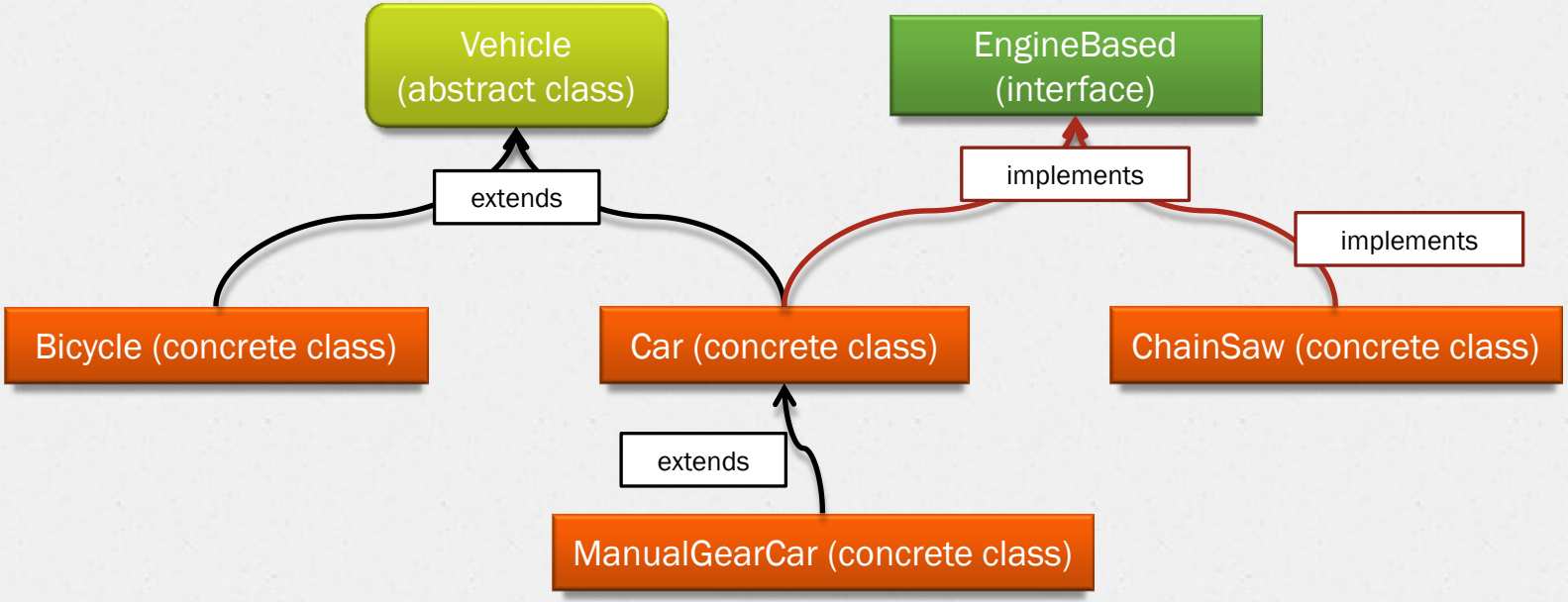
- Write an interface for API-only classes.

```
public interface EngineBased {
            void startEngine();
            void stopEngine();
}
public class Car extends Vehicle implements EngineBased {
            private String licenseNumber;
            public String getLicenseNumber() {
                        return licenseNumber;
            }
            public void stop() {
                        hitBreaksPedal();
            }
            public void startEngine() { switchKey(); }
            public void stopEngine() { switchKey(); }
}
public class Bicycles extends Vehicle { (no change)
```

Interfaces (cont'd)

```
public interface EngineBased {
          void startEngine();
          void stopEngine();
}
public class ChainSaw implements EngineBased {
          void startEngine() {
                    pullCordReallyStrong();
          }
          void stopEngine() {
                    clickStopButton();
          }
}
```

Class Diagram



Using

Casting & polymorphism

```
public class CellularPhone {
        public void dial(String number) {
                // some implementation
        }
        public void recharge() {
                // connect to recharger
        }
}
```

```
public class IPhone extends CellularPhone {
        // overriding a method
        public void dial(String number) {
                // iPhone specific implementation
        }
        // extending a method
        public void runApplication(String appName) {
                // start the application
        }
}
```

Polymorphism

- Consider the following method:

```
public void do(CellularPhone phone) {
        phone.recharge();
        phone.dial("1-800-JAVA");
        phone.runApp("TextIt");
}
```

- These represent the 3 basic scenarios:
  - **`recharge()`** is implemented only in **`CellularPhone`**.
  - **`dial()`** is implemented in **`CellularPhone`** and overridden in **`IPhone`**.
  - **`runApp()`** is implemented only in **`Iphone`**.

Always think!

- Will it compile?
  - What is the reference type?
  - Does it have such a method?
- What will run?
  - What is the current object in memory, referenced by the reference variable?
  - Does it implement the method?
  - Examples of different object types in runtime:

```
do(new CellularPhone());
do(new IPhone());
```

phone.recharge()

```
public void do(CellularPhone phone) {
        phone.recharge();
```

- Reminder: `recharge()` is implemented only in `CellularPhone`.
- Will it compile?
  - What is the reference type? `CellularPhone`
  - Does it have such a method? Yes.
  - Hurray! It will compile!
- What will run?
  - What is the current object in memory? Does it implement the method?
    - If the object is of type `CellularPhone`, then yes.
    - If the object is of type `IPhone`, then no.
  - The implementation in `CellularPhone.recharge()`

phone.dial()

```
public void do(CellularPhone phone) {
        phone.dial("1-800-JAVA");
```

- Reminder: **dial()** is implemented in **CellularPhone** and overridden in **IPhone**.
- Will it compile?
  - What is the reference type? **CellularPhone**
  - Does it have such a method? Yes.
  - Hurray! It will compile!
- What will run?
  - What is the current object in memory? Does it implement the method?
    - If the object is of type **CellularPhone**, then yes.
    - If the object is of type **IPhone**, then yes.
  - The implementation of the object in memory will run.

phone.runApp()

```
public void do(CellularPhone phone) {
        phone.runApp("TextIt");
```

- Reminder: `runApp()` is implemented only in `IPhone`.
- Will it compile?
  - What is the reference type? **CellularPhone**
  - Does it have such a method? No.
  - The code will not compile.
- The only way to compile it is by telling the compiler "look at phone as a reference to IPhone".
- It's done by casting:

```
public void do(CellularPhone phone) {
        ((IPhone)phone).runApp("TextIt");
```

casting

```
public void do(CellularPhone phone) {
        ((IPhone)phone).runApp("TextIt");
```

- Will it compile?
  - What is the reference type? **IPhone**
  - Does it have such a method? Yes.
  - Hurray! It will compile!
- What will run?
  - What is the current object in memory? Does it implement the method?
    - If the object is of type **IPhone**, then yes.
    - If the object is of type **CellularPhone**, then no, and we get a **ClassCastException**.
  - We avoid such an exception by making sure the object in memory is of type **IPhone**.

instanceof

```
public void do(CellularPhone phone) {
          if (phone instanceof IPhone) {
                    ((IPhone)phone).runApp("TextIt");
          }
```

- What will run?
  - What is the current object in memory? Does it implement the method?
    - If the object is of type **IPhone**, then yes.
    - If the object is of type **CellularPhone**, then the condition will be false, and the line will be ignored.

You must admit that inheritance & polymorphism is beautiful and romantic

If you admit it, welcome to the geeks club ☺