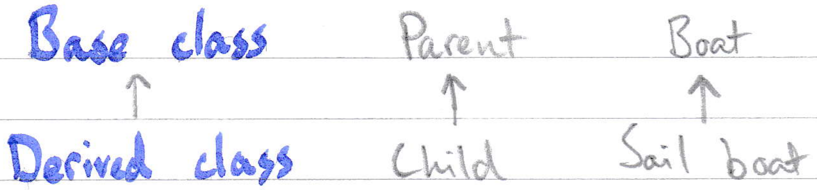
Relationships - 3 relationships that can be set up in OOP

1) Inheritance

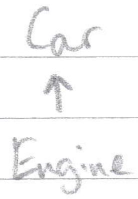
"is - a" relationship between classes. Lets you add extra features to a base class



To set up, use keyword "extends"

2) Containment

"has - a" relationship



To set up, place object into private, then that class has an object of that type.

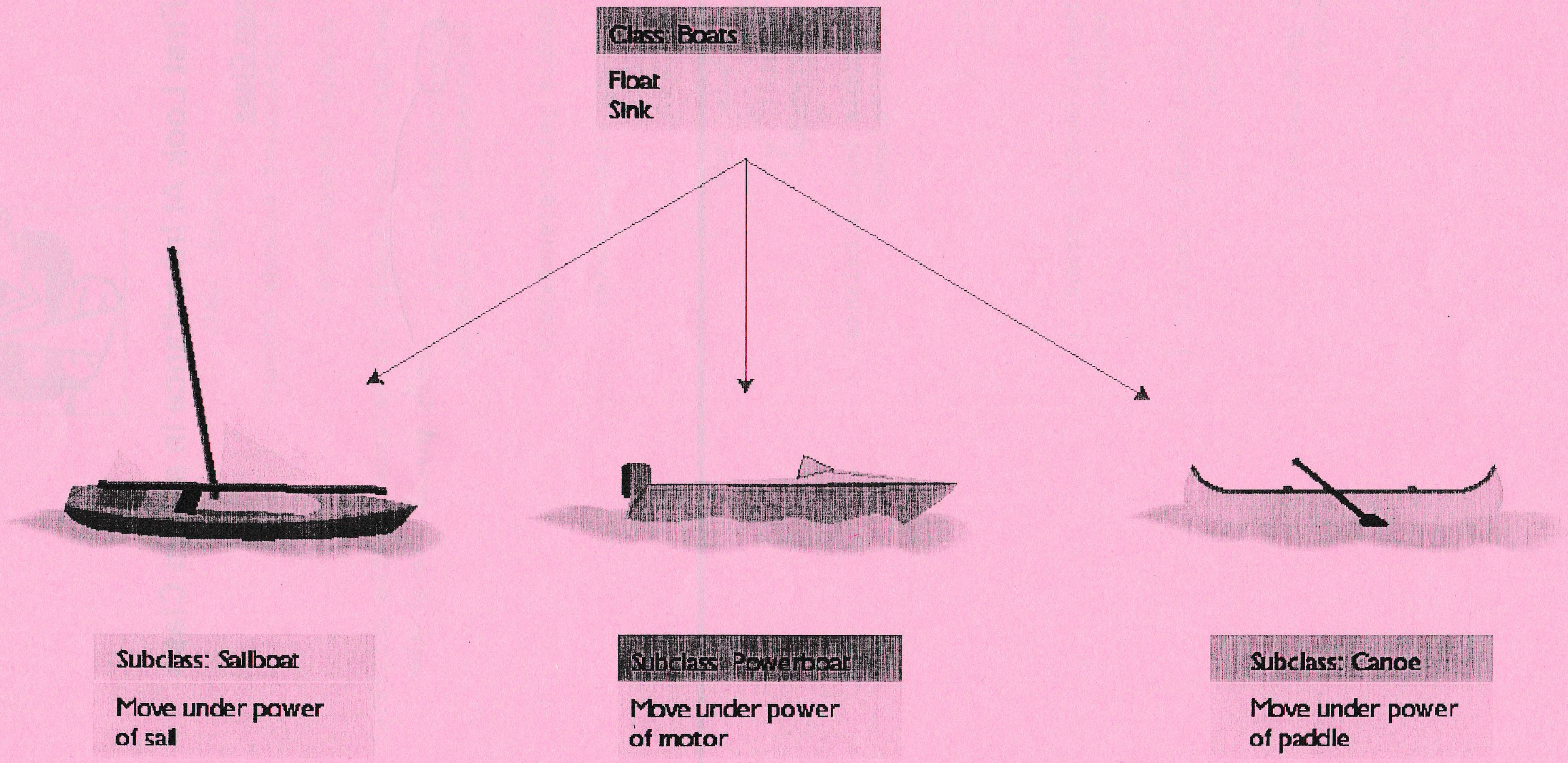
```
class Car {
   private engine e;
}
```

3) Polymorphism

Allows an object of a derived type to "morph" at Run Time when passed into a base type

To set up, send a child object (one derived from base object) at Run Time and having it morph into a correct one

Inheritance in Object-Oriented



Class: Boats

Float
Sink

Subclass: Sailboat

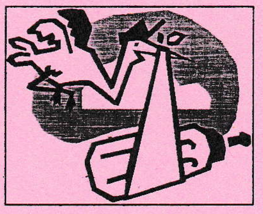
Move under power
of sail

Subclass: Powerboat

Move under power
of motor

Subclass: Canoe

Move under power
of paddle



Our First Look At Inheritance in a Java Class

The SavingsAccount Class

- derived class

```
public class SavingsAccount extends Account
{
        private double interestRate;

        public SavingsAccount(double initBalance, double initRate)
        {
                super(initBalance);
                interestRate = initRate;
        }

        public double getInterestRate()
        {
                return interestRate;
        }
}
```

calls parent constructor

The Account Class

- base class

```
public class Account {
  private double balance;

  public Account(double initialBalance) {
    balance = initialBalance;
  }

  public Account() {
    balance = 0.0;
  }

  public void deposit(double amount) {
    balance += amount;
  }

  public void withdraw(double amount) {
    balance -= amount;
  }

  public double getBalance() {
    return balance;
  }

  public void close() {
    balance = 0.0;
  }
}
```

Polymorphism Examples

The MyAccounts Program

```
public class MyAccounts
{
        public static void main(String[] args)
        {
                Account rAcct = new Account(100.0);
                printMe(rAcct);

                SavingsAccount sAcct = new SavingsAccount(100.0, 0.5);
                printMe(sAcct);

        }

        private static void printMe(Account a)
        {
                System.out.println(a);
        }
}
```

morphs into savings account

The Substitution Principle

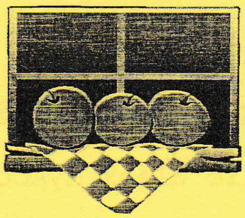
To allow polymorphism, Java has a rule that might be called the ***Substitution Principle:***

> *An instance of a subclass can take the place of an instance of any of its superclasses.*

Polymorphism Examples

The Fruity Program

4 classes in 1 file



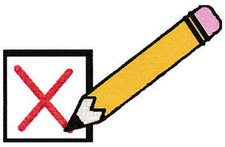
```
class Fruit
{
     public void print() {
               System.out.println("Fruit");
          }
}

class Orange extends Fruit
{
      public void print() {  //overriding
               System.out.println("Orange");
          }
      public void printTwo() {
              super.print(); // calls base class print
          }
}

class Mandarin extends Orange
{
      public void print() {
              System.out.println("Mandarin");
          }
};


public class Fruity
{
        public static void main(String[] args)
        {
          try{
                Fruit a = new Mandarin(); polymorphism (*poly)
                a.print(); ← output: Mandarin
                a = new Orange(); *poly2
                a.print(); ← output: Orange

                Fruit f = new Fruit();
                Mandarin m = new Mandarin();
                f = m; *poly3

                if (f instanceof Orange)
                {
                        Orange o = (Orange) f; *poly4   to get compile
                        o.printTwo(); ← output: Fruit
                }
          }
          catch (ClassCastException e)
          {
                System.err.println(e.getMessage());
          }
        }
}
```

derived class rewrites a method / with same name from its base class

Object
↑
Fruit
↑
printTwo method — Orange
↑
Mandarin

is-a relationship

Practice with Inheritance & Polymorphism



Valid or invalid code fragments?

Account = parent

1)

```
Account acct = new Account(100.00); ✓
SavingsAccount savingsAcct = acct; ×
```

Invalid

child

↑ child cannot go into parent

2)

```
SavingsAccount savingsAcct = new SavingsAccount(100.00, 5.0); ✓
Account acct = (Account) savingsAcct; ✓
```

Valid

polymorphism

↗ cast

if (Account) is not there, won't compile

3)

```
Account acct = new Account(100.00);
SavingsAccount savingsAcct = (SavingsAccount) acct;
```

Invalid

parent == child, will compile

4)

```
Account acct = new SavingsAccount(100.00, 5.0);
SavingsAccount savingsAcct = acct;
```

Invalid

↑ doesn't compile because no cast

5)

```
SavingsAccount savingsAcct = new Account(100.00);
Account acct = savingsAcct;
```

Invalid

no cast, doesn't work

6)

```
Account acct = new SavingsAccount(100.00, 5.0);
SavingsAccount savingsAcct = (SavingsAccount) acct;
```

Valid

7)

```
SavingsAccount savingsAcct = new Account(100.00);
Account acct = (Account) savingsAcct;
```

Invalid

Compiler error - when data doesn't match