Last Class

- Parameter passing
  - Call-by-value vs call-by-reference
- Inheritance
  - Public, package, protected, private
  - Overriding
  - Overloading
- Constru....

Constructors

- Every class comes with a default constructor with no parameters
- Additional constructors with any number of parameters can be defined.

  doing so turns off the default constructor -- DANGER
- If a subclass has a custom constructor

  A constructor of the superclass is called

  Then statements in the subclass' constructor are executed
- A subclass may explicitly access its super class's constructors through the key word **`super`**

Example -- Constructors

```
public class A
{
   String s="???";
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      A a = new A();
      System.out.println(a);
   }}
```

```
public class B
{
   String s="???";
   public B(String ss) { s=ss; }
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      B b = new B();
      System.out.println(b);
   }
}
```

What is the result of A and B?
Do they even compile?

More Constructors

```
public class BB
{
   String s="???";
   public BB(String ss) { s=ss; }
   public String toString()
   { return s; }
   public static void main(String[] args)
   {
      BB b = new BB("q");
      System.out.println(b);
   }
}
```

```
public class C extends BB {
   public static void main(String[] args) {
      C c = new C();
   }
}
```

Problem: C does not compile

error message:

symbol : constructor B()

location: class B

Interpretation:

C is looking for no arg constructor in B

Fix:

1. Put "public B() {}" in B
2. Put "public C() { super(null); } in C

The granddaddy class

- In Java, a class may extend only one class (single inheritance/single parenting)
- As we have seen, a class can have many children. This leads to a tree structure.
- At the top of the inheritance tree is the class **`Object`** – all classes extend **`Object`** by default

```
class A extends Object
```

References and casting

- Reference variable of type superclass can be used to refer to a subclass, but not vice versa
- If must be done, use explicit casting – similar to numeric type conversion
- Can be tested using **`instanceof`**

  e.g. `If (a instanceof B) { ... }`

Casting

```
public class Person {
   protected String name="";
   protected Person bff=null;
   public Person(String s) {
      name=s;}
   public void setBFF(Person p) {
      bff = p; }
   public String toString() {
      return "name="+name+"  bff="+bff; }
   public String getName() { return name; }
   public static void main(String[] args) {
      Person p1 = new Person("jane");
      Person p2 = new Person("joan");
      Person p3 = new Person("jean");
      p1.setBFF(p2);
      p2.setBFF(p3);
      p3.setBFF(p1);
      System.out.println(p1);
   }}
```

```
public class Adult extends Person {
   public Adult(String n) { super(n); }
   public String toString() {
      return getName() + bff.getName();
   }
   public static void main(String[] args) {
      Adult a1 = new Adult("john");
      Adult a2 = new Adult("jack");
      a1.setBFF(a2);
      a2.setBFF((Person) a1);
      System.out.println(a1 instanceof Person);
      System.out.println(a2 instanceof Adult);
      System.out.println(a1);
   }
}
```

Thinking about Algorithms

Permuting a list

Alg1: Swapping through list

```
for (int k=0; k<size; k++)
      tgt[k]=k;
for (int k=0; k<tgt.length; k++)
{
      int pl = (int)(Math.random()*tgt.length);
      int t = tgt[k];
      tgt[k]=tgt[pl];
      tgt[pl]=t;
}
```

Good: fast

Bad: randomness??? How do you test this?

More Algorithm Thoughts

What went wrong?

Maybe it had something to do with using “k” as one of the things swapped?? If yes, then replace k with a random

```
for (int k=0; k<size; k++)
    tgt[k]=k;
      for (int k=0; k<tgt.length; k++)
          {
             int p1 = (int)(Math.random()*tgt.length);
             int p2 = (int)(Math.random()*tgt.length);
             int t = tgt[p2];
             tgt[p2]=tgt[p1];
             tgt[p1]=t;
          }
```

Good: fixes problem with randomness??

Bad: slower

Yet more thoughts

maybe swapping was a bad idea?

1. Create a list of numbers (1-52)
2. for k from 1 to 52
   select a number from the list from step 1
   call it the kth item in new list
   remove selected number from list of step 1

Good: maybe gives nice randomness

Bad: array implementation of remove step may be very slow

Making classes

- Goal
  - Take the deck of cards from the lab and make it object oriented
  - Steps
    - Decide of a preliminary set of objects
    - Within each object
      - Decide what data the object contains
      - Decide how other objects can interact with instances of instances of the object
        - i.e., what is public?
        - This is often referred to as the API or “Application Programming Interface”