Announcements

- P6 posted. Due Thurs, 5/5, at 6pm
- Prelim 3: pick up at Carpenter during consulting hrs only, in general 2:30-6pm daily (see Staff link of website for exact times)

Prelim 3

- Median 76
- Mean 72.9; Standard Deviation 15.5
- A rough mapping of grades:
  - Q (A-like) 88-100
  - R (B-like) 73-85
  - S (C-like) 58-70

Prelim 3

- Q1: definition, method call ☺ ☺
- Q2: algorithm (loop, keeping track of previous values) ☺
- Q3: algorithm (while loop, conditionals) ☺
- Q4: OOP ☺

Previous Lecture:
- Extending a class
- protected visibility
- Calling the superclass’ constructor

Today’s Lecture:
- Overriding methods
- Using super to access methods in superclass
- What is “polymorphism”?

Reading:
- Sec 8.2, Sec 9.1, 9.2

TrickDice is a subclass of Dice

```java
class Dice {
    private int top;
    private int sides;
    public Dice(…) {...}
    public void roll() {...}
    public String toString(){...}
    public int getTop() {...}
    public int getSides() {...}
}

class TrickDice extends Dice {
    private int weightedSide;
    private int weight;
    public TrickDice(…) {...}
    public void roll() {...}
    public String toString() {...}
    public int getWSide() {...}
    public int getWeight() {...}
}
```

Overriding methods

- Subclass can override definition of inherited method
- New method in subclass must have same signature as superclass (but has different method body)
- Which method gets used??
  - The object that is used to invoke a method determines which version is used
- Method declared to be final cannot be overridden
- Do not confuse overriding with overloading!
Accessing members in superclass

- From constructor in subclass, call superclass' constructor
- Access superclass' version of a overridden method. E.g.:
  
  ```java
  super.toString()
  ```

**static methods & variables**

- Do not re-declare static components!
- Same rules for inheritance (accessibility) with respect to visibility modifiers
- Static method: implicitly final
- Static variable: same memory space as superclass

**Important ideas in inheritance**

- Single inheritance
- Keep common features as high in the hierarchy as reasonably possible
- Use the superclass' features as much as possible
- "Inherited" ⇒ "can be accessed as though declared locally"
  
  (private variables in superclass exists in subclasses; they just cannot be accessed directly)
- Inherited features are continually passed down the line
- Use different hierarchies for different problems

**Polymorphism**

- “Have many forms”
  
  - A polymorphic reference refers to different objects (related through inheritance) at different times

Suppose class `Plane` extends `Vehicle`

```java
Vehicle mover; //a Vehicle reference
Plane flyer;   //a Plane reference
mover= new Vehicle(...);
flyer= new Plane(...);
// A plane is a vehicle
mover= new Plane(...);
mover= flyer;
// A vehicle is not a plane
flyer= new Vehicle(...); //invalid
```

**Another polymorphic example**

```java
Vehicle[] mover = new Vehicle[5];
mover[0]= new Vehicle(...);
mover[1]= new Plane(...);
mover[2]= new Plane(...);
mover[3]= mover[1];
```
Accessing methods/variables through polymorphic references

Ask two questions:
1. What determines whether a method/variable can be accessed?
   - reference type
2. For an overridden method, what determines which version gets invoked?
   - object type

Client code:

```java
V x = new W();
System.out.println(x.num1);
System.out.println(x.num2);  //invalid
x.vmethod();  //invalid
x.wmethod();  //invalid
System.out.println(((W)x).num2);
((W)x).wmethod();
```

Client code:

```java
V x;  // x references type V or its subtype
String s = "Which type, V or W? ";
char input = Keyboard.readChar();
if (input == 'V')
x = new V();
else
x = new W();
System.out.println(x.num1);
System.out.println(x.num2);  //invalid
x.vmethod();
x.wmethod();  //invalid
```

Accessing overridden methods through polymorphic references

- The type of the object determines which version of the method gets invoked
- Class Vehicle has method toString that class Plane overrides:

```java
Vehicle v1 = new Vehicle(...);
Vehicle v2 = new Plane(...);
System.out.println(v1);  //Vehicle’s version
System.out.println(v2);  //Plane’s version
```