administrivia...
-assignment 1 due today at midnight

-assignment 2 is out
  -due next Thursday at midnight
  -requires pair programming!

-labs start on Monday

-clickers ($#%*!!!!!!)
last time...
inheritance
public class Triangle{
    String color;
    double area;
}

public class Circle{
    String color;
    double area;
}

public class Rectangle{
    String color;
    double area;
}

public class Circle{
    String color;
    double area;
}

WHAT IF I WANT TO REDDEFINE COLOR AS AN INTEGER ARRAY (R,G,B)?

WHAT IF I WANT TO GIVE EACH SHAPE AN OUTLINE COLOR?

WHAT CAN I DO?
public class Triangle{
    String color;
    double area;
}

public class Circle{
    String color;
    double area;
}

public class Rectangle{
    String color;
    double area;
}

public class Circle{
    String color;
    double area;
}

WHAT IF I WANT TO REDFINE COLOR AS AN INTEGER ARRAY (R,G,B)?

WHAT IF I WANT TO GIVE EACH SHAPE AN OUTLINE COLOR?

WHAT CAN I DO?

extends
public class Shape{
    String color;
    double area;
}

called a **base class** (or superclass)

public class Triangle extends Shape{
}

public class Circle extends Shape{
}

public class Rectangle extends Shape{
}

public class Circle extends Shape{
}

inherit all public fields and methods of Shape
suppose you are making a video game about skiing

```java
public class Ski{
    public void turn();
}

public class AlpineSki extends Ski{
    // override
    public void turn(){
        // how to turn on alpine skis
    }
}

public class TelemarkSki extends Ski{
    // override
    public void turn(){
        // how to turn on tele skis
    }
}
```
suppose you are making a video game about skiing

WITHOUT INHERITANCE:

switch(skier.ski_type)
{
    case ALPINE:
        turnAlpine();
    break;
    case TELEMARK:
        turnTelemark();
    break;
    ...
}

WITH INHERITANCE:

skier.ski.turn();
**polymorphism** is a fancy word for automatically determining an object’s type at runtime

-the most specific type possible is used

```java
Shape s1 = new Circle();
Shape s2 = new Triangle();

s1.getArea();
s2.getArea();
```
-**polymorphism** is a fancy word for automatically determining an object’s type at runtime

-the most specific type possible is used

```java
Shape s1 = new Circle();
Shape s2 = new Triangle();

s1.getArea();
WHAT TYPE IS s1 TREATED AS?
s2.getArea();
```
**polymorphism** is a fancy word for automatically determining an object’s type at runtime

the most specific type possible is used

```java
Shape s1 = new Circle();
Shape s2 = new Triangle();

s1.getArea();
s2.getArea();
```

WHAT TYPE IS `s1` TREATED AS?
WHAT TYPE IS `s2` TREATED AS?
**Polymorphism** is a fancy word for automatically determining an object’s type at runtime

- **the most specific type possible is used**

```java
Shape s1 = new Circle();
Shape s2 = new Triangle();

s1.getArea();
s2.getArea();
```

**What type is** `s1` **treated as?**

**What type is** `s2` **treated as?**

- **Suppose** `Triangle` **does not override** `toString()`

```java
s2.toString();
```
**polymorphism** is a fancy word for automatically determining an object’s type at runtime

-the most specific type possible is used

```java
Shape s1 = new Circle();
Shape s2 = new Triangle();
```

s1.getArea();  \hspace{2cm} \text{WHAT TYPE IS s1 TREATED AS?}

s2.getArea();  \hspace{2cm} \text{WHAT TYPE IS s2 TREATED AS?}

-suppose Triangle does not override toString()

s2.toString();  \hspace{2cm} \text{WHAT TYPE IS s2 TREATED AS?}
- A class with at least one `abstract` method is an `abstract` class.

- Derived classes **MUST** implement `abstract` methods.

- `abstract` classes cannot be instantiated.

```java
Shape s = new Shape();
Shape s = new Triangle();
```

Which of these is illegal?

- `abstract` classes are **ONLY** designated as base classes.
-an **interface** is the ultimate abstract class
  -every method is **abstract**
  -can contain only **public static final** fields
  -declared with the **interface** keyword instead of **class**

-derived classes use keyword **implements** instead of **extends**

-subclasses can implement multiple interfaces, but can only extend one base class
today...
- generic programming
- generic placeholder
- why generics
- primitive types and generics
- generic static methods
- function objects
generic programming
- Suppose we want a data structure that just contains “things”

- We want it to:
  - Automatically grow if it gets full
  - Be able to remove items from it
  - Be able to add items to it
suppose we want a data structure that just contains “things”

we want it to:
- automatically grow if it gets full
- be able to remove items from it
- be able to add items to it

will an array work?

```java
Shape[] shape_array = new Shape[5];
```
-how about an ArrayList?
- here’s what the code might look like:

```java
public class ArrayList {
    Shape storage[];
    int capacity, numItems;

    public void addItem(Shape item) {
        /*some code*/
    }

    public void autoGrow() {
        /*some code*/
    }
}
```
-how about an ArrayList?
-Here's what the code might look like:

```java
public class ArrayList {
    Shape storage[];
    int capacity, numItems;

    public void addItem(Shape item) {
       /*some code*/
    }

    public void autoGrow() {
       /*some code*/
    }
}
```

What's the problem with this?
-this is why we always see <> associated with ArrayList

ArrayList<Shape> list = new ArrayList<Shape>();

-ArrayList is a generic class — we can create any version of it that we want

-generic programming: algorithms are written in terms of types to-be-specified-later
  -algorithms instantiated when needed for specific types defined by parameters
- here’s what the code actually looks like:

```java
public class ArrayList<T> {
    T storage[];
    int capacity, numItems;

    public void add(T item)
    {
        ... 
    }
}
```

- the placeholder `T` is replaced with the real type when you instantiate an `ArrayList` with `<>

- `T` can be used as a type anywhere in `ArrayList` class
generic placeholder
WHAT IS THE DYNAMIC TYPE OF T?

ArrayList<Shape>
WHAT IS THE DYNAMIC TYPE OF \( T \)?

ArrayList<Shape>

ArrayList<ClassThatArrayListDoesntKnowAbout>
**generic placeholder <>**

**WHAT IS THE DYNAMIC TYPE OF T?**

`ArrayList<Shape>`

`ArrayList<ClassThatArrayListDoesntKnowAbout>`

-the generic placeholder type is **VERY** specific
  - `ArrayList<Triangle>` is not an `ArrayList<Shape>`, even though `Triangle` is a `Shape`!

-`ArrayList<type>` is only **EXACTLY** an `ArrayList<type>`, **regardless of type’s heritage**
inheritance and generics

-example:

```java
public void doStuff(ArrayList<Shape>) {...}

ArrayList<Triangle> tri_list;
ArrayList<Shape> shape_list;

doStuff(tri_list);  // ILLEGAL
doStuff(shape_list); // OK
```

-we can still add **Triangles** to **shape_list**

-restriction applies only to the generic object itself
Java has a way around the restriction: the **wildcard** placeholder `?`.

`<? extends Shape>` refers to **Shape** or anything that extends **Shape**
- **Shape**, **Triangle**, **Circle**, ...
Java has a way around the restriction: the wildcard placeholder `<? extends Shape>` refers to `Shape` or anything that extends `Shape`

- `Shape`, `Triangle`, `Circle`, ...

WHAT TYPES CAN BE USED HERE?
Java has a way around the restriction: the **wildcard placeholder**

```java
<? extends Shape>
```

refers to `Shape` or anything that extends `Shape`

- `Shape`, `Triangle`, `Circle`, ...

**WHAT TYPES CAN BE USED HERE?**

```java
<? super Circle>
```
-Java has a way around the restriction: the **wildcard placeholder** `?`?

-`<? extends Shape>` refers to Shape or anything that extends Shape
  -Shape, Triangle, Circle, ...

**WHAT TYPES CAN BE USED HERE?**

`<? super Circle>`

`<?>`
Java has a way around the restriction: the **wildcard** placeholder?

- `<? extends Shape>` refers to `Shape` or anything that extends `Shape`
  - `Shape`, `Triangle`, `Circle`, ...

**WHAT TYPES CAN BE USED HERE?**

`<? super Circle>`

`<??>` **IS THIS A GOOD IDEA?**
why generics?
-everything in Java is an Object
-so, why not just make all data structures hold Objects?
-everything in Java is an `Object`
  - so, why not just make all data structures hold `Objects`?

-`generics` allow for type-checking at compile time instead of run-time
-everything in Java is an Object
  -so, why not just make all data structures hold Objects?

-generics allow for type-checking at compile time instead of run-time

-can detect type mismatch BEFORE your code runs
BEFORE GENERICS:

ArrayList l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // crash
BEFORE GENERICS:

ArrayList l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // crash

ALTERNATIVE:

ArrayList<String> l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // compile error
BEFORE GENERICS:

```java
ArrayList l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // crash
```

ALTERNATIVE:

```java
ArrayList<String> l;
l.add(new String("hi"));
Shape i = (Shape)l.get(0); // compile error
```

COMPILATION TIME ERRORS ARE ALWAYS BETTER THAN RUN-TIME!
primitive types and generics
-generics only work with reference types
  -no int, char, float, double, ...

-what if we need an ArrayList of ints?

-Java has “wrapper” classes
  -Integer, Float, Double
  -these are reference types containing a single primitive...
  -...and methods to access it
    -intValue(), doubleValue()
Java will automatically insert the appropriate code to convert between primitive/reference

```java
ArrayList<Integer> l;

l.add(5);

int i = l.get(n);
```

EQUIVALENT TO

```java
l.add(new Integer(5));

int i = l.get(n).intValue();
```

EQUIVALENT TO

```java
int i = l.get(n).intValue();
```
questions...
questions...

WHAT TYPES ARE INCLUDED IN <? super Triangle>?
1. Shape
2. Triangle, Circle, Rectangle, Square
3. Triangle, Shape, Object
questions...

WHAT TYPES ARE INCLUDED IN super Triangle>
1. Shape
2. Triangle, Circle, Rectangle, Square
3. Triangle, Shape, Object

WHAT TYPES ARE INCLUDED IN <Shape>
1. Shape
2. Triangle, Circle, Rectangle, Square
3. both 1 and 2
generic static methods
- **static** methods can have their own generic types

- declare the generic type before the return type:
  
  ```java
  public static <T> boolean doWork(...) {...}
  ```

- we can refer to `T` as a type within that method only!
- **static** methods can have their own generic types

- declare the generic type before the return type:

  ```java
  public static <T> boolean doWork(...) {...}
  ``

- we can refer to T as a type within that method only!

- example:
- **static methods can have their own generic types**

- **declare the generic type before the return type:**
  public static <T> boolean doWork(...) {...}

- **we can refer to T as a type within that method only!**

- **example:**

  ```java
  public static <T> boolean contains(T[] array, T item) {
      for(int i=0; i < array.length; i++)
          if(array[i].equals(item))
              return true;
  
      return false;
  }
  ```
function objects
-suppose we want a generic sorting function
-and we want it to be able to sort ANY type…
-suppose we want a generic sorting function
-and we want it to be able to sort ANY type…
-what can we do?
suppose we want a generic sorting function
and we want it to be able to sort ANY type…
what can we do?
what do we need to be able to do?
- suppose we want a generic sorting function
  - and we want it to be able to sort ANY type...
  - what can we do?
  - what do we need to be able to do?
    **DECIDE WHICH ITEM IS LARGER**
Comparable interface

```java
public interface Comparable<T> {
    public int compareTo(T item);
}
```

- defines a natural ordering *(in fact, it is contractually obligated to!)*
  - String, Integer, ... all implement Comparable

- what if we want a different ordering? or to order Shapes? or to order Strings based on length?
function objects

-a function object is an object that defines a single method

-example:
  -a Comparator has a single method: compare
    -takes two arguments
    -decides which one is greater

-we write a sorting function that takes a Comparator

WHAT DOES THIS ALLOW US TO DO?
Comparator interface

public interface Comparator<T> {
    int compare(T left, T right);
}

-returns a number <0 if left < right

-returns a 0 if they are equal

-returns a number >0 if left > right
next time...
-reading
  - chapters 5 & 6

-homework
  - assignment 1 due today at midnight
  - assignment 2 due next Thursday at midnight
    - must complete with a partner!

-lab on Monday
  - timing