**Data Types & Operations**

**Overview**
- **Data type – structure of information**
  - Existing/User-defined
    - Existing include int, String, double, boolean, Rectangle
    - User defined are user written classes whose variables and methods describe the structure
  - Primitives/Reference
    - Primitives store only a single value (and no methods)
    - References can store many values and methods

**Data Types**
- **Primitives**
  - double – real numbers
  - int – integers
  - boolean – true or false
  - char – single letter, number, space, or punctuation
  - long – larger range of integers
  - float – smaller precision real numbers

- **References (Objects)**
  - String – in java.lang
  - Rectangle – in java.awt
  - Greeter – in Horstmann
  - Purse – in Horstmann
  - ChangeMaker – going over this today
  - PolarCoordinate – going over this today
Operations

- **Operators**
  - Most are binary (having two operands)
  - Some are unary (having one operand)
  - compute a value

- **Methods**
  - static (do not require an instance, also known as class methods)
  - instance
  - may or may not compute a value (non-void or void)
  - can have zero or more parameters

Methods

- **Static**
  - Math.sin – sine
  - Math.cos – cosine
  - Math.abs – absolute value

- **Instance**
  - sayHello of Greeter
  - getDollars of ChangeMaker
  - getMagnitude of PolarCoordinate
  - equals of PolarCoordinate

ChangeMaker

Converts dollars and cents to actual bills and coins.

Example:
$37.43 can be
1 $20 bill
1 $10 bill
1 $5 bill
2 $1 bills
1 quarter
1 dime
1 nickel
3 pennies

ChangeMaker.java

```java
public class ChangeMaker {
    int dollars;
    int cents;

    public ChangeMaker() {
        this(0, 0);
    }

    public ChangeMaker(int d, int c) {
        dollars = d;
        cents = c;
    }
}
```

int means integer valued (no fractional parts)

Constructors initialize the instance variables dollars and cents

Assignment operators
public class ChangeMaker
{
    int dollars;
    int cents;
    public ChangeMaker()
    {
        this(0, 0);
    }
    public ChangeMaker(int d, int c)
    {
        dollars=d;
        cents=c;
    }

    public void addCents(int c)
    {
        cents = cents + c;
        dollars = dollars + cents / 100;
        cents = cents % 100;
    }
    public void addDollars(int d)
    {
        dollars = dollars + d;
    }

    public int getDollars()
    {
        return dollars;
    }
    public int getcents()
    {
        return cents;
    }
}

Calls the second constructor from the first.

Mutator methods modify instance variables

+ adds
/ does division
In this case integer division because cents is an integer and 100 is an integer
Order of operations gives precedence over +

Integer division truncates – by dropping the fractional part. 120/100 is 1 not 1.2

Mutator methods modify instance variables

Accessor methods give access to the instance variables
```java
public int get20s()
{
    return dollars / 20;
}

public int get10s()
{
    return (dollars % 20) / 10;
}
```

Integer division because dollars and 20 are both integers.

Integer division truncates – by dropping the fractional part. 50/20 is 2 not 2.5

If dollars is 50, what is 50%20?
What is (50%20)/10?
How many $10 bills are needed when trying to give $50 in the largest denominations possible? (with $20 bills the largest)

% gets the remainder after integer division. For example 50%20 is 10 7%5 is 2 5%7 is 5

Parenthesis may be used to indicate the desired order of computation.

If dollars is 50, what is the result of calling get5s()?
If dollars is 37 what is the result of calling get5s()?
Do the get5s() and get1s() instance methods compute the correct answer? Why?
The toString() instance method converts instance variable values into a meaningful String. For example, if dollars is 5 and cents is 14 the toString() Method returns the String “$5.14”.

Why does computing the return value of the toString() method depend on the value of cents?

Driver for testing the ChangeMaker class

Calls the constructor

public ChangeMaker(int d, int c) {
    dollars = d;
    cents = c;
}
```java
public static void main(String[] argv) {
    ChangeMaker money=new ChangeMaker(37, 43);
    System.out.println(money + " has: ");
    System.out.println(money.get20s() + " 20 dollar bills ");
    System.out.println(money.get10s() + " 10 dollar bills ");
    System.out.println(money.get5s() + " 5 dollar bills ");
    System.out.println(money.get1s() + " 1 dollar bills ");
    System.out.println(money.getQuarters() + " quarters ");
    System.out.println(money.getDimes() + " dimes ");
    System.out.println(money.getNickels() + " nickels ");
    System.out.println(money.getPennies() + " pennies ");
}
```

Automatically calls the `toString()` method

```java
public String toString() {
    if(cents<10)
        return "$"+dollars+.0+cents;
    else
        return "$"+dollars+"."+cents;
}
```

calls the `get20s()` method

```java
public int get20s() {
    return dollars/20;
}
```

```java
public static void main(String[] argv) {
    ChangeMaker money=new ChangeMaker(37, 43);
    System.out.println(money + " has: ");
    System.out.println(money.get20s() + " 20 dollar bills ");
    System.out.println(money.get10s() + " 10 dollar bills ");
    System.out.println(money.get5s() + " 5 dollar bills ");
    System.out.println(money.get1s() + " 1 dollar bills ");
    System.out.println(money.getQuarters() + " quarters ");
    System.out.println(money.getDimes() + " dimes ");
    System.out.println(money.getNickels() + " nickels ");
    System.out.println(money.getPennies() + " pennies ");
}
```

main outputs:
$37.43 has:
1 20 dollar bills
1 10 dollar bills
1 5 dollar bills
2 1 dollar bills
1 quarters
1 dimes
1 nickels
3 pennies

Rectangular coordinates are broken down into horizontal and vertical components pairs, or just (x, y)
Polar coordinates are broken down into magnitude and angle from the origin or (r, theta)
The conversion exists (r, theta) in polar is equivalent to (r cosine(theta), r sine(theta))
sometimes polar coordinates are easier to deal with, especially when doing rotations.
PolarCoordinate.java provides routines for converting between polar and cartesian coordinates
import java.awt.*;
import java.awt.geom.*;

public class PolarCoordinate {
    private static final double EPSILON = 1e-10;
    private double magnitude;
    private double angle;

    public PolarCoordinate() {
        this(0, 0);
    }

    public PolarCoordinate(double pm, double pa) {
        setMagnitude(pm);
        setAngle(pa);
    }

    public Point getPoint() {
        int x = (int)(magnitude * Math.cos(angle));
        int y = (int)(magnitude * Math.sin(angle));
        return new Point(x, y);
    }

    public Point2D.Double getPoint2D() {
        double x = magnitude * Math.cos(angle);
        double y = magnitude * Math.sin(angle);
        return new Point2D.Double(x, y);
    }

    public double getMagnitude() {
        return magnitude;
    }

    public double getAngle() {
        return angle;
    }

    public void setMagnitude(double pm) {
        magnitude = pm;
    }

    public void setAngle(double pa) {
        angle = pa;
    }

    static final is used to denote constants
    Note the all capitals
    Shorthand for scientific notation 1x10^-10
    Constructors can call instance methods
    Casting the results which are doubles into ints which truncates
    Static methods of the Math class
    Which are accessor methods?
    Which are mutator methods?
```
public boolean equals(Object object) {
    if(object instanceof PolarCoordinate) {
        PolarCoordinate polar=(PolarCoordinate)object;
        if(Math.abs(polar.angle-angle)<EPSILON &&
            Math.abs(polar.magnitude-magnitude)<EPSILON) {
            return true;
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

Focus on this part of the code.

Floating point arithmetic incurs small errors which can cause two numbers which would theoretically be the same to be slightly different. If comparing floating point numbers for equivalence, check to see if they are roughly the same rather than exactly equal.

```
if(Math.abs(polar.angle-angle)<EPSILON &&
    Math.abs(polar.magnitude-magnitude)<EPSILON) {
    return true;
} else {
    return false;
}
```

abs is for absolute value

Why is the absolute difference needed?

All arguments are computed before Calling the method.

```
if(Math.abs(polar.angle-angle)<EPSILON &&
    Math.abs(polar.magnitude-magnitude)<EPSILON) {
    return true;
} else {
    return false;
}
```

Each PolarCoordinate object has its own angle and magnitude. If there is no object name before an instance variable, the assumed object is **this** (which means the current object). *more explanation about this later.*
public static double convertToDegrees(double radians) {
    return radians * 180 / Math.PI;
}

public static double convertToRadians(double degrees) {
    return degrees * Math.PI / 180;
}

static methods do not access or modify any instance variables
Notice angle and magnitude are not used in these methods.

// does floating point division because Math.PI is a double
precision floating point number (so are radians and degrees)

public static double convertToRadians(double degrees) {
    return degrees * Math.PI / 180;
}

Floating point division
retains the fractional part of the division

if(one==two) {
    System.out.println("one and two reference the same object");
} else {
    System.out.println("one and two reference different objects");
}

if(one.equals(two)) {
    System.out.println("one and two have the same contents");
} else {
    System.out.println("one and two have different contents");
}

With Objects, == is true if the two sides reference
the same Object

With Objects, the equals method checks the
Contents of the objects for equality

Driver for PolarCoordinate.java (continued on next slide)
Summary

- Data types
  - Primitives
  - References
- Operations
  - Operators
  - Methods

Reminders

- Should have read Chapters 1-3 of Head First Java
- Intro to Video Game Package Monday
- Graphics on Wednesday
- Homework 1 due on Wednesday