6.092

# Lecture 7

Good programming skills

Collections

Exceptions

iapjava-staff@mit.edu

# Assignment 6 Review

```
public abstract class BouncingDevice {

    int x; // used to belong to BouncingBox

    int y;

    public BouncingDevice(int startX, int startY,
  Color startColor) { // constructor

    }

    public void animate () {

        // move animation code here.

    }

    public void moveInDirection (int dx, int dy) {

    }

}
```

# Assignment 6 Review

```
public class BouncingBall extends BouncingDevice {
    public BouncingBall (int x, int y, Color c) {
        super (x, y, c);
    }

    public void draw (Graphics2D surface) {
        // drawing code specific to a ball
    }
}
```

# Assignment 6 Review

- Move as much as you can to BouncingDevice
- Abstract method draw: excellent idea!
  ```
  public abstract void draw ();
  ```
- `moveInDirection()` at the wrong place again!
- Please submit **all** your Java files
- You cannot call **super** outside a constructor!

# Assignment 6 Review

```
public class BouncingBall extends BouncingDevice {
    public BouncingBall (int x, int y, Color c) {
        super (x, y, c);
    }

    public void draw (Graphics2D surface) {
        super.animate (); // NO!!
    }
}
```

# Assignment 6 Review

```
public class BouncingBall extends BouncingDevice {

    public BouncingBall (int startX, int startY, Color startColor){
        super (startX, startY, startColor);
    }

    public void draw (Graphics2D surface){
        surface.setColor(color);
         surface.fillOval(x - SIZE/2, y - SIZE/2, SIZE, SIZE);
         surface.setColor(Color.BLACK);
         surface.setStroke(new BasicStroke(3.0f));
         surface.drawOval(x - SIZE/2, y - SIZE/2, SIZE, SIZE);
         animate();
         moveInDirection(xDirection, yDirection); // NO!
         }
    }
```

# Refresher

## Intro/Overview

– compilation, execution• Java Basics:
– Structure & Syntax, Variables, Types, & Operators

## Control Flow

– Methods & Conditionals, Loops & Arrays

## Object-oriented Programming (OOP):

– Objects & Classes
– Inheritance & Abstraction:

## Classes, Abstract Classes & Interfaces

– Encapsulation

## Brief Intro to Software Design

# Outline

- Good programming skills (II)
- Collections
- Exceptions

# Good programming skills

- **Use meaningful variable and method names.**
- **Indent your code.**
- **What else?**

# Good programming skills

- **Use abstraction to avoid duplicating code.**

```
//GOOD
public abstract class BouncingDevice {
   public int x, y;

   public abstract void draw ();
}

public abstract class BouncingBall {
   public void draw () {
      // ...
   }
}
```

# Good programming skills

**Use abstraction to avoid duplicating code.**

```
//BAD
public abstract class BouncingBall {
   public int x, y;
   public void draw () {
      // does stuff
   }
}

public abstract class BouncingBox {
   public int x, y;
   public void draw () {
      // does the same stuff
   }
}
```

# Good programming skills

- **Comment your code, but not too much!**

```
//GOOD
public abstract class BouncingDevice {
   public int x, y; // device position

   /* draw the device on the image */
   public abstract void draw ();
}
```

# Good programming skills

- **Comment your code, but not too much!**

```
//BAD
public abstract class BouncingDevice {
   public int x, y;
   public abstract void draw ();
}
```

```
//BAD
public abstract class BouncingDevice {
    /* We define a bouncing device class; it contains a position x,y
and has a drawing method draw() that draws the object on the image; Do
not forget the food for the cat on Wednesday; Oh by the way, this
project is due next week, I need to send an email to the instructors
telling them that I will be late... */
    public int x, y;
    public abstract void draw ();
}
```

# Good programming skills

- **Have a main() method in each class for unit testing.**

```
//GOOD
public abstract class BouncingDevice {
   public int x, y; // device position

   public static void main (String[] args) {

   }
}
```

# Good programming skills

- **Start small. Focus on the core capabilities first (skip the details). Do not over-anticipate.**

```
// BAD
public class BouncingDevice {
  public int switchingColor;
}
```

# Good programming skills

- **Use meaningful variable and method names.**
- **Indent your code.**
- **Use abstraction.**
- **Comment your code.**
- **Use main() for unit testing.**

# Why should I write nice code?

- Save yourself some time when you read the code 10 weeks/month/years later.
- Help your friends understand your code in a team's project.
- Help your instructor/TA give you a good grade.
- Make debugging faster by not duplicating code.

# Outline

- Good programming skills (II)
- **Collections**
- Exceptions

# Collections

**The Problem with arrays:**

- ✗ Not resizable
- ✗ Not useful for creating mappings between objects (requires at least three arrays)
- ✗ Not useful for keeping track of duplicate objects
- ✗ Not useful for constant-time operations

# Collections

**The Solution: Collections**

- ✔ Allow to create dynamic groupings (Set), orderings (List), and mappings (Map) between objects
- ✔ Mirror mathematical constructs
- ✔ Are automatically resized to fit new members
- ✔ Live in java.util package

# Collections

- Example of a collection: ArrayList.
- But there is much more!

```
ArrayList<BouncingBox> boxes = new
  ArrayList<BouncingBox>();

BouncingBox b = new BouncingBox (200, 50, Color.RED);

boxes.add(b);

BouncingBox d = boxes.get(0);
```

# Collections

- **Collection**
  - generic container, most of the framework implements this
- **List**
  - stores multiple items in an explicit order (rep. elts allowed)
  - **ArrayList**, **LinkedList**, etc.
- **Set**
  - stores unique items in no particular order
  - **HashSet**, **SortedSet**, etc.
- **Map**
  - stores unordered key-value pairs (like a dictionary; keys are unique)
  - **HashMap**, **Hastable**, **TreeMap**, etc.

# Collections

- Basic useful methods:
  - add
  - addAll
  - remove
  - clear
  - isEmpty
  - size (not length!)
  - toArray

- See API for more + usage!

# Collections Generics

- Collections can hold objects of different runtime types, though we generally don't and shouldn't

- Generics allow one to specify the type of the elements in a Collection
  - Avoids messy casting
  - Enables us to use more than just plain Object

# Collections Generics

- Instantiation:

```
Person[] p = new Person[10];
ArrayList<Person> al = new ArrayList<Person>();
```

- Iteration:

```
Set<Person> s = new HashSet<Person>();
Iterator<Person> i = s.iterator();
while (i.hasnext()) { p = i.next(); }
```

**or**

```
for (Person p : s) { p.doSomething(); }
```

# Outline

- Good programming skills (II)
- Collections
- **Exceptions**

# Exceptions

- A way to tell when something goes wrong in a method call
- When an error happens, an Exception object is thrown
- Useful for debugging & control flow

# Exception Types

- Common types of Exceptions
  - RuntimeExceptions
    - NullPointerException
    - ClassCastException
    - ArrayIndexOutOfBoundsException
    - Etc.
  - Other Exceptions

# Throwing an Exception

- To declare that you throw an exception:

```
public int pop(int size) throws EmptyStackException
  {


   if (size == 0) {
       throw new EmptyStackException();
  }
  size--;
  return size;
}
```

# Catching an Exception

- Using a method that throws an Exception
  - try it
  - If it doesn't work, it will throw its Exception
  - Then you must catch the exception
  - You can catch multiple Exception types

# Catching an Exception

```
try {
      pop(0);
} catch (EmptyStackException e) {
     System.err.println("Blah”);
     throw new SampleException(e);
} catch (IOException e) {
     System.err.println("Blah again!”);
}
```

# Assignment 7: YourFace

- Build a (simple) social network in Java!