Lecture 8
Math and Making Decisions

Outline

• Abstract methods and classes
• Arithmetic operations – java.lang.Math
• Static methods and static variables
• Constants – values that never change
• Decision making: boolean algebra, if-else statements and the switch statement

Recall: Inheritance Example

• Remember the Race program from the last lecture
  - our transportation options include Convertible, CS15Mobile, and Van
  - each of these extend Car and inherit methods from it
  - we can represent this in an Inheritance/Interface Diagram
Abstract Methods and Classes (1/6)

- What if we wanted to seat all of the passengers in the car?
- CSIMobile, Convertible, and Van all have different numbers of seats
  - they will all have different implementations of the same method

abstract Methods and Classes (2/6)

- We declare a method abstract in a superclass when the subclass can’t really re-use any implementation the superclass might provide – no code-reuse
- In this case, we know that all Cars should loadPassengers, but each subclass will loadPassengers very differently
- abstract method is declared in superclass, but not defined – it is up to subclasses farther down hierarchy to provide their own implementations
- Thus superclass specifies a contractual obligation to its subclasses – just like an interface does to its implementors
Here, we've modified Car to make it an abstract class: a class with at least one abstract method.

We declare both Car and its loadPassengers method abstract: if one of a class's methods is abstract, the class itself must also be declared abstract.

An abstract method is only declared by the superclass, not defined – thus use semicolon after declaration instead of curly braces.

```java
class Car {
    private Racer driver;
    public Car(Racer driver) {
        this.driver = driver;
    }
    public abstract void loadPassengers();
}
```

How do you load Passengers?
- Every Passenger must be told to sit in a specific Seat in a physical Car.
- SeatGenerator has methods that returns a Seat in a specific logical position.

```java
class Passenger {
    public Passenger() {
        // code elided
    }
    public void sit(Seat seat) {
        // code elided
    }
}
class SeatGenerator {
    public SeatGenerator() {
        // code elided
    }
    public Seat getShotgun() {
        // code elided
    }
    public Seat getBackLeft() {
        // code elided
    }
    public Seat getBackCenter() {
        // code elided
    }
    public Seat getBackRight() {
        // code elided
    }
    public Seat getMiddleLeft() {
        // code elided
    }
    public Seat getMiddleRight() {
        // code elided
    }
}
```

All concrete subclasses of Car override by providing a concrete implementation for Car's abstract loadPassengers() method.

As usual, method signature and return type must match the one that Car declared.

```java
class Convertible extends Car {
    @Override
    public void loadPassengers() {
        SeatGenerator seatGen = new SeatGenerator();
        this.passenger1.sit(seatGen.getShotgun());
    }
}
class CS15Mobile extends Car {
    @Override
    public void loadPassengers() {
        SeatGenerator seatGen = new SeatGenerator();
        this.passenger1.sit(seatGen.getShotgun());
        this.passenger2.sit(seatGen.getBackLeft());
        this.passenger3.sit(seatGen.getBackCenter());
    }
}
class Van extends Car {
    @Override
    public void loadPassengers() {
        SeatGenerator seatGen = new SeatGenerator();
        this.passenger1.sit(seatGen.getMiddleLeft());
        this.passenger2.sit(seatGen.getMiddleRight());
        this.passenger3.sit(seatGen.getBackLeft());
        // more code elided
    }
}
```
abstract Methods and Classes (6/6)

- abstract classes cannot be instantiated!
  - this makes sense – shouldn’t be able to just instantiate a generic Car, since it has no code to loadPassengers()
  - instead, provide implementation of loadPassengers() in concrete subclass, and instantiate subclass
- Subclasses: at any level in inheritance hierarchy can make an abstract method concrete by providing implementation
  - it’s common to have multiple consecutive levels of abstract classes before reaching a concrete class
- Even though an abstract class can’t be instantiated, its constructor must still be invoked via super() by a subclass
  - because only the superclass knows about (and therefore only it can initialize) its own instance variables

So.. What’s the difference?

- You might be wondering: what’s the difference between abstract classes and interfaces?
  - abstract classes:
    - can define instance variables
    - can define a mix of concrete and abstract methods
    - you can only inherit from one class
  - Interfaces:
    - cannot define any instance variables/concrete methods
    - has only undefined methods (no instance variables)
    - you can implement multiple interfaces

Summary of Inheritance and Polymorphism

- Inheritance models very similar classes
  - factor out all similar capabilities into a generic superclass
  - superclasses can
    - declare and define methods
    - declare abstract methods
  - subclasses can
    - inherit methods from a superclass
    - define their own specialized methods
    - completely/partially override an inherited method
- Polymorphism allows programmers to reference instances of a subclass as their superclass
  - Inheritance, Interfaces, and Polymorphism take generic programming to the max – more in later lecture

Note: Java, like most programming languages, is evolving. In Java 8, interfaces and abstract classes are even closer in that you can have concrete methods in interfaces. We will not make use of this in CS15.
Outline

- Abstract methods and classes
- Arithmetic operations – java.lang.Math
- Static methods and static variables
- Constants – values that never change
- Decision making: boolean algebra, if-else statements and the switch statement

Review: Basic Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Equivalent Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>a += 5;</td>
<td>a = a + 5;</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>a -= 5;</td>
<td>a = a - 5;</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>a *= 5;</td>
<td>a = a * 5;</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>a /= 5;</td>
<td>a = a / 5;</td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
<td>a %= 5;</td>
<td>a = a % 5;</td>
</tr>
</tbody>
</table>

Basic Arithmetic Operators: Shorthand

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Equivalent Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>add and reassign</td>
<td>a += 5;</td>
<td>a = a + 5;</td>
</tr>
<tr>
<td>-=</td>
<td>subtract and reassign</td>
<td>a -= 5;</td>
<td>a = a - 5;</td>
</tr>
<tr>
<td>*=</td>
<td>multiply and reassign</td>
<td>a *= 5;</td>
<td>a = a * 5;</td>
</tr>
<tr>
<td>/=</td>
<td>divide and reassign</td>
<td>a /= 5;</td>
<td>a = a / 5;</td>
</tr>
<tr>
<td>%=</td>
<td>take remainder and reassign</td>
<td>a %= 5;</td>
<td>a = a % 5;</td>
</tr>
</tbody>
</table>
Unary Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>negate</td>
<td>b = -b; // negates b</td>
</tr>
<tr>
<td>++</td>
<td>increment</td>
<td>b++; // equivalent to: b = b + 1;</td>
</tr>
<tr>
<td>--</td>
<td>decrement</td>
<td>b--; // equivalent to: b = b - 1;</td>
</tr>
</tbody>
</table>

Increment and Decrement Operators

- `++` and `--` can be applied before (prefix) or after (postfix) the operand
- `i++` and `++i` will both increment variable `i`
- `i++` assigns, then increments
- `++i` increments, then assigns

Postfix example:
```java
int i = 10;
int j = i++;
// j becomes 10, i becomes 11
```

Prefix example:
```java
int i = 10;
int j = ++i;
// i becomes 11, j becomes 11
```

java.lang.Math

- Extremely useful "utility" class, part of core Java libraries
- Provides methods for basic numeric operations
  - absolute value: `abs(double a)`
  - exponential: `exp(double a, double b)`
  - natural and base 10 logarithm: `log(double a), log10(double a)`
  - square root: `sqrt(double a)`
  - trigonometric functions: `cos(double a), sin(double a)`...
  - random number generation: `random()` returns random number from 0.0(inclusive) to 1.0(exclusive)
- for more check out: [https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html)
Outline

- Abstract methods and classes
- Arithmetic operations – java.lang.Math
- Static methods and static variables
- Constants – values that never change
- Decision making: boolean algebra, if-else statements and the switch statement

static Methods

- All of java.lang.Math's methods are declared static
- Example: the method that returns the absolute value of an integer is declared below
  - public static int abs(int a) {...}
- A static method belongs to a class, rather than an instance of the class
  - it cannot access instance variables, whose values may differ from instance to instance
    - but can have local variables, e.g., temps

Calling a static Method

- static methods are invoked on the class, not on an instance:
  - int absoluteValue = Math.abs(-7);
- That means we can use all of Math's static methods without ever instantiating it

Note: You won't need to write any static methods of your own in CS15, but you'll be using Math's static methods in future assignments
TopHat Question

playerCounter is an instance of the Survivor class. Which is the correct way to call this static method:

```java
public static int playersLeft() {...}
```

A. int contestants = Instance.playersLeft();
B. int contestants = Survivor.playersLeft(static);
C. int contestants = SurvivorInstance.playersLeft(static);
D. int contestants = Survivor.playersLeft();
E. int contestants = playerCounter.playersLeft();

---

**static Variables**

- Progression in scope:
  - **local** variables are known in a single method
  - **instance** variables are known to all methods of a class
  - **static** instance variables are known to all instances of a class
- Each instance of a class has the same instance variables but typically with different values for those properties
- If instead you want all instances of a class to share the same value for a variable, declare it **static** — this is not very common
- Each time any instance changes the value of a static variable, all instances have access to that new value

```java
public class Player {
    private static int players = 0;
    public Player () {
        this.players++;
    }
    public static int getNumPlayers() {
        return this.players;
    }
}
```

---

**static Variables: Simple Example**

- players starts out with a value of 0
- Each time a new instance of Player is created, players is incremented by 1
- Get current value at any point by calling Player.getNumPlayers();
  - each instance of Player will have the same value of players
- static methods can use static and local variables — but not instance variables
Abstract methods and classes

Arithmetic operations – java.lang.Math

Static methods and static variables

Constants – values that never change

Decision making: boolean algebra, if-else statements and the switch statement

Outline

• Abstract methods and classes
• Arithmetic operations – java.lang.Math
• Static methods and static variables
• Constants – values that never change
• Decision making: boolean algebra, if-else statements and the switch statement

Constants

• Constants are used to represent values which never change (e.g. Pi, speed of light, etc.) – very common!
• Keywords used when defining a constant:
  o public: value should be available for use by anyone (unlike private instance variables and local variables)
  o static: all instances of the class share one value
  o final: value cannot be reassigned
  o naming convention for constants is all caps with underscores between words: LIGHT_SPEED

Constants: Example (1/2)

• Useful to bundle a bunch of constants for your application in a “utility” class (like Math), with useful methods using those constants; both constants and methods will be then declared static:

  public abstract class Physics {
      // speed of light (Units: hundred million m/s)
      public static final double LIGHT_SPEED = 2.998;
      // constructor elided
      public static double getDistanceTraveled(double numSeconds) {
          return (LIGHT_SPEED * numSeconds);
      }
  }
Constants: Example (2/2)

• Always use constants when possible
  - literal numbers, except for 0 and 1, should rarely appear in your code
  - makes code readable, easier to alter
• Also called symbolic constants – should have descriptive names
• If many classes use same constants, make separate utility class, like `Physics`
• A constants utility class should never be instantiated, so it should be declared abstract

```java
public abstract class Physics {
    // speed of light (Units: hundred million m/s)
    public static final double LIGHT_SPEED = 2.998;
    // we can add more constants if we want
}
```

We can access this constant from a method in another class in our program like this: `Physics.LIGHT_SPEED`

Example:
```
spaceship.setSpeed(Physics.LIGHT_SPEED)
```

TopHat Question

Which of the following constants is defined correctly?

A. `public static final int PLAYER_AGE;`
B. `public static final int PLAYER_AGE = 20;`
C. `public static int final PLAYER_AGE = 20;`
D. `private static final int PLAYER_AGE = 20;`

Workout (1/6)

• Contestant Boston Rob decides to prepare for Survivor by trying Andy’s `super calf workout`—let’s model it!
• Depending on his weight and time of his workout, he will gain a certain amount of calf muscle
• Our Head TAs calculated that his effort is the weight times his workout time
• Muscle gained equals one tenth of the square root of his effort
Workout (2/6)

- `WorkoutConstants` class keeps track of important constants in our calculation

```java
public abstract class WorkoutConstants {
    // Weight
    public static final double START_WEIGHT = 150;

    // Don't want him to gain more muscle than this
    public static final double MAX_WEIGHT = 200;
}
```

Workout (3/6)

- Rob keeps track of instance variable `weight`
- `weight` initialized in constructor to starting weight defined in `WorkoutConstants`

```java
import java.lang.Math;

public class Rob {
    private double weight;

    public Rob() {
        this.weight = WorkoutConstants.START_WEIGHT;
    }
}
```

Workout (4/6)

- Rob’s `gainMuscle` method changes his weight depending on the amount of time he works out

```java
import java.lang.Math;

public class Rob {
    private double weight;

    public Rob() {
        this.weight = WorkoutConstants.START_WEIGHT;
    }

    public void gainMuscle(double workoutTime) {
        // code elided
    }
}
```
Workout (5/6)

- First, effort is computed
- Second, muscleGained is calculated according to the formula
- Math.sqrt is a static method from java.lang.Math that computes the square root of a value
- Increment the weight with the muscle gained

```java
import java.lang.Math;
public class Rob {
    private double weight;
    public Rob() {
        this.weight = WorkoutConstants.START_WEIGHT;
    }
    public void gainMuscle(double workoutTime) {
        double effort = workoutTime * this.weight;
        double muscleGained = (1/10) * Math.sqrt(effort);
        this.weight += muscleGained;
    }
}
```

Workout (6/6)

- Now fill in `calfWorkout()`
- Rob will only work out if weight is not already above maximum WEIGHT
- How can we check if condition is met?
  - Introducing... boolean's and if's!
    - seen booleans in Pong assignment but let's formally introduce them

```java
import java.lang.Math;
public class Rob {
    private double weight;
    public Rob() {
        this.weight = WorkoutConstants.START_WEIGHT;
    }
    public void gainMuscle(double workoutTime) {
        double effort = workoutTime * this.weight;
        double muscleGained = (1/10) * Math.sqrt(effort);
        this.weight += muscleGained;
    }
    public void calfWorkout() {
        // code to workout!
    }
}
```

Outline

- Abstract methods and classes
- Arithmetic operations – java.lang.Math
- Static methods and static variables
- Constants – values that never change
- Decision making: boolean algebra, if-else statements, and the switch statement
### booleans

- British logician George Boole (1815-1864) wanted to improve on Aristotelian (formal) logic, e.g., modus ponens, rule of inference:
  
  "All men are mortal. Socrates is a man, therefore..."

- **boolean** (named after Boole) is simplest Java base type

- A **boolean** variable can have value **true** or **false**

- Example initialization:

  ```java
  boolean foo = true;
  boolean bar = false;
  ```

  The terms foo, bar, etc. are often used as placeholder names in computer programming or computer-related documentation: derived from FUBAR, WWII slang.

### Relational Operators

- Can compare numerical expressions with **relational operators**

- Full expression evaluates to a **boolean**: either **true** or **false**

- Examples:

  ```java
  boolean b1 = (3 > 2);
  boolean b2 = (5 == 5);
  int x = 8;
  boolean b3 = (x <= 6);
  ```

  - **b1** and **b2** are **true**, **b3** is **false**

### Comparing References

- Can use **==** and **!=** to see if two references point to the same instance, or not

- What three values are printed to the console in this example?

  1. **false**: d1 and d2 are not equal
  2. **true**: d1 and d2 refer to the same instance
  3. **true**: d1 != d2 is false, so foo is true (since foo = !(false))
TopHat Question
Which of the following will print false?

```java
public class TestClass {
    // constructor elided
    public void compareReferences() {
        Student s1 = new Student();
        Student s2 = new Student();
        boolean sameStudent = (s1 == s2);
        A. System.out.println(sameStudent);
        s1 = s2;
        sameStudent = (s1 == s2);
        System.out.println(sameStudent);
        boolean student1Exists = (s1 != null);
        System.out.println(student1Exists);
    }
}
```

A.  
B.  
C.  

• if statements allow us to make decisions based on value of a boolean expression
• Syntax:
  ```java
  if (boolean expression) {
      // code to be executed if expression is true
  }
  ```
• If boolean expression is true, code in body of if statement is executed. If false, code in body skipped
• Either way, Java compiler continues on with rest of method

if Statement: Flow Chart

---

40/75

41

42/75

42

10/5/21
### if Statements: Examples

```java
int x = 6;
if (x == 5) {
    // code to execute if x is 5
}
```

Not executed

```java
int y = 9;
//more code elided
```

```java
if (y > 7) {
    // code to execute if y is greater than 7
}
```

Executed

---

### Logical Operators: And, Or, Not (1/2)

- Logical operators `&&` ("and") and `||` ("or") can be used to combine two boolean expressions
  - `<expression a> && <expression b>` evaluates to true only if both expressions are true
  - `<expression a> || <expression b>` evaluates to true if at least one expression is true
- Logical operator `!` ("not") negates a boolean expression
- Logical operator `^` ("exclusive or") returns true if either `a` or `b` is true but not both

---

### Logical Operators: And, Or, Not (2/2)

- To represent the values a logical operator may take, a truth table is used

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp;&amp; B</th>
<th>A</th>
<th>B</th>
<th>A^B</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
TopHat Question

Which if clause statement will run if Rob is not eliminated and there is a challenge tomorrow? (The variables below are of type boolean)

A. if(!robEliminated && !challengeTomorrow){…}
B. if(!robEliminated && challengeTomorrow){…}
C. if(robEliminated && !challengeTomorrow){…}
D. if(robEliminated && challengeTomorrow){…}

if Statements: More Examples

• Should always take one of two forms:
  - if (<boolean expression>)
  - if (!<boolean expression>)

• Never do this (inefficient):
  - if (<boolean expression> == true)
  - if (<boolean expression> == false)

• Be careful! It’s easy to mistakenly use = (assignment operator) instead of == (comparator)

if-else (1/2)

• If we want to do two different things depending on whether the boolean expression is true or false, we can use an else clause

  Syntax:
  
  if (<boolean expression>) {
    // code executed if expression is true
  } else {
    // code executed if expression is false
  }
**if-else: Flow Chart**

- **Previous Statements**
- **Is condition true?**
  - No
  - Execute else clause
  - Execute rest of method

**if-else (2/2)**

- Can use `if-else` to fill in the `calfWorkout` method
- If Rob’s weight is less than the maximum weight when the method is called, he gains muscle
- Otherwise, he stops and joins Survivor!
- Does this code limit the final calf weight to `MAX_WEIGHT`?

```java
import java.lang.Math;
public class Rob {
    private double weight;
    // constructor elided
    public void gainMuscle(double workoutTime) {
        double effort = workoutTime * this.weight;
        double muscleGained = (1/10) * Math.sqrt(effort);
        this.weight += muscleGained;
    }
    public void calfWorkout() {
        if (this.weight < WorkoutConstants.MAX_WEIGHT) {
            this.gainMuscle(60.0); // work out for 60 minutes!
        } else {
            // this method defined elsewhere in the code
            this.joinSurvivor();
        }
    }
}
```

**Complex if-else Statements**

- If `boolean expression 1` is true, block 1 is executed and blocks 2 and 3 are skipped
- If `boolean expression 1` is false and `boolean expression 2` is true, block 2 is executed and blocks 1 and 3 are skipped
- If both expressions are false, block 3 is executed and blocks 1 and 2 are skipped
Nested if Statements

```java
// variables and methods defined elsewhere
if (cs15Student.hasProject()) {
    if (cs15Student.hasInitiative()) {
        cs15Student.workOnProject();
    } else {
        cs15Student.doChallenge();
    }
}
```

TopHat Question

Which print statement will be printed out?

```java
int x = 10;
if (x < 10) {
    if ((x+10) > 15) {
        System.out.println("case A");
    } else {
        System.out.println("case B");
    }
} else if (x <= 15) {
    if ((x+2) > 13) {
        System.out.println("case C");
    } else {
        System.out.println("case D");
    }
} else {
    System.out.println("case E");
}
```

Short-Circuiting (1/2)

- What is the value of `n` after the code to the right has executed?
  - `n` is still 1
  - Why?

```java
int n = 1;
if ((n < 0) && (n++ == 2)) {
    // code to be executed if expression is true
}
System.out.println(n);
```
Short-Circuiting (2/2)

- Beware of short-circuiting!
- If Java already knows what the full expression will evaluate to after evaluating left argument, no need to evaluate right argument
  - &&: if left argument of conditional evaluates to False, right argument not evaluated
  - ||: if left argument evaluates to true, right argument not evaluated

```java
int n = 1;
if ((n == 1) || (n == 2)) {
    // code to be executed if expression is true
}
int n = 1;
if (false && (n++ == 2)) {
    // code to be executed if expression is true
}
System.out.println(n);
```

“Side-effect”ing

- Updating a variable inside a conditional is not good coding style; it makes code confusing and hard to read
- Keep in mind short-circuiting if you ever call a method that might have a “side effect” inside a conditional – here the first if will leave n incremented, second not

```java
int n = 1;
if ((n++ == 2) && false) {
    // code to be executed if expression is true
}
System.out.println(n);
int n = 1;
if (false && (n++ == 2)) {
    // code to be executed if expression is true
}
System.out.println(n);
```

switch Statements (1/2)

- If want to do something different for every possible value of a an integer variable, have two options:
  - use a lot of else-if:
    ```java
    if (myInteger == 0) {
        // do something...
    } else if (myInteger == 1) {
        // do something else...
    } else if (myInteger == 2) {
        // do something else...
    } else if (myInteger == 3) {
        // do something else...
    } else ...
    // last case
    }
    ```
  - better solution: use a switch statement:
    ```java
    switch (myInteger) {
        case 0:
            // code to be executed if myInteger == 0
            break;
        case 1:
            // code to be executed if myInteger == 1
            break;
        case 2:
            // code to be executed if myInteger == 2
            break;
        case 3:
            // code to be executed if myInteger == 3
            break;
        default:
            // last case
    }
    ```
switch Statements (2/2)

Syntax:
```java
switch (variable) {
    case value:
        // do something
        break;
    case other value:
        // do something else
        break;
    default:
        // take default action
        break;
}
```

Rules:
- `<variable>` usually an integer; char and enum (discussed later) also possible
- values have to be mutually exclusive
- If `default` is not specified, Java compiler will not do anything for unspecified values
- `break` indicates the end of a `case` - skips to end of switch statement (if you forget `break`, the code in next case will execute)

switch Example (1/6)

- Let’s make a `BandanaCreator` class that produces different colored bandanas for contestants using a switch statement
- The bandana is chosen by weighted distribution (more orange, red, brown, and fewer blue, green, yellow)
- `BandanaCreator` generates random values using `Math`
- Based on random value, creates and returns a bandana of a particular type

switch Example (2/6)

- To generate a random value, we use static method `random` from `java.lang.Math`
- `random` returns a `double` between 0.0 (inclusive) and 1.0 (exclusive)
- This line returns a random `int 0-9` by multiplying the value returned by `random` by 10 and casting the result to an `int`
- Casting is a way of changing the type of an object to another specified type. Casting from a `double` to `int` truncates your `double`!
We initialize myBandana to null, and switch on the random value we've generated.

```java
public class BandanaCreator{
    public Bandana generateBandana(){
        int rand_int = (int) (Math.random() * 10);
        Bandana myBandana = null;
        switch(rand_int){
            case 0: case 1:
                myBandana = new Bandana(Color.ORANGE);
                break;
            case 2: case 3: case 4:
                myBandana = new Bandana(Color.RED);
                break;
        }
    }
}
```

- Bandana takes in an instance of javafx.scene.paint.Color as a parameter of its constructor (needs to know what color it is)
- Once you import javafx.scene.paint.Color, you only need to say, for example, Color.ORANGE to name a color of type Color
- If random value turns out to be 0 or 1, instantiate an orange Bandana and assign it to myBandana
- break breaks us out of switch statement

If our random value is 2, 3, or 4, we instantiate a red Bandana and assign it to myBandana.

Color.RED is another constant of type Color – check out Javadocs for javafx.scene.paint.Color!
switch Example (6/6)

```java
public class BandanaCreator {
    // constructor elided
    public Bandana generateBandana() {
        int rand_int = (int) (Math.random() * 10);
        Bandana myBandana = null;
        switch (rand_int) {
            case 0: case 1:
                myBandana = new Bandana(Color.ORANGE);
                break;
            case 2: case 3: case 4:
                myBandana = new Bandana(Color.RED);
                break;
            // cases 5, 6, and 7 elided.
            // they are green, blue, yellow.
            default:
                myBandana = new Bandana(Color.BROWN);
                break;
        }
        return myBandana;
    }
}
```

TopHat Question

Which of the following switch statements is correct?

A. ```java
   int rand = (int) (Math.random() * 10);
   Island island = null;
   switch (rand) {
       case 0: case 1: case 2: case 3:
           island = new Island("Maui");
           break;
       case 4: case 5: case 6: case 7:
           island = new Island("Bali");
           break;
       default:
           island = new Island("Fiji");
           break;
   }
```

B. ```java
   Place place = Place.random();
   Island island = null;
   switch (place) {
       case Maui:
           island = new Island("Maui");
           break;
       case Bali:
           island = new Island("Bali");
           break;
       default:
           island = new Island("Fiji");
           break;
   }
```

C. ```java
   int rand = (int) (Math.random() * 10);
   Island island = null;
   switch (rand) {
       case 0: case 1: case 2: case 3:
           island = new Island("Maui");
           break;
       case 4: case 5: case 6: case 7:
           island = new Island("Bali");
           break;
       default:
           island = new Island("Fiji");
           break;
   }
```

That's It!

Important Concepts:
- static methods and static variables
- Constants
- booleans
- Making decisions with if, if-else, switch
Announcements

- TicTacToe late handin due today at 11:59 PM
- FruitNinja (handout and help slides) released tomorrow
  - Early handin: 10/19 (+2 bonus points)
  - On-time handin: 10/19
  - Late handin: 10/14 (0 for late handin, but 3 late days to use throughout semester)
- Debugging Hours start tomorrow 10/06
- Polymorphism section this week
  - email your section TA mini-assignment on time
- Course calendar
  - Course Syllabus
  - Course Schedule

Topics in Socially Responsible Computing
Drones and Robotics

Robotics

- A machine that is controlled by computer(s) and can carry out complex actions (semi-) automatically
  - Not all robots are anthropomorphic!
- Types:
  - Pre-programmed
  - Autonomous
  - Humanoid
  - Tele-operated
  - Augmenting
Unmanned Automatic Vehicles (Drones)

- Deliver packages (to layperson)
- Useful for delivering medicine and supplies
  - Increase equity of medical care
  - Transport across unreliable terrain with minimal risk of contamination
    - Zipline services 2,500 hospitals and health clinics across Rwanda and Ghana during the pandemic
    - Zipline also delivers to remote villages in rural Chile
  - Vaccine distribution for those living in “pharmacy deserts”

- Documentation and activism
  - 2016-2017 used by protestors at Standing Rock, ND
    - Revealed proximity of pipeline to tribe’s water
    - Recorded police brutality and protected counter-narratives from law enforcement and media
  - May 2020: photographer George Steinmetz exposed NYC COVID “mass grave”
    - Hart Island: where city buried COVID victims whose bodies were not claimed for private burial
      - Graves dug by Rikers Island jail inmates
      - NYPD confiscated drone + detained George, resulting in legal battle
  - High resolution images of dangerous or difficult to access places

- Surveillance
  - Drones used for public and private surveillance
    - Ex. EU used drones in 2015 to surveil protestors in Scotland

- Warfare
  - UAVs have been used by US military since ~2001 to surveil and perform unmanned strikes
    - combo.
    - Distance enables error, dehumanization of targets
    - Up to 25% of victims are civilians
    - ~25% of civilian victims are children

Photo: TechCrunch

Photo: TechCrunch

Photo: ACLU

Photo: ACLU
Robotics @ Brown

- Faculty
  - George Konidaris
  - Michael Littman
  - Stefanie Tellex
    - Flying robots curriculum for high schoolers

- Groups
  - Humanity Centered Robotics Initiative
  - Humans 2 Robots Lab
  - Intelligent Robot Lab
  - RLAB (reinforcement learning and adaptive behavior) group

Back to Facebook

- Facebook products “harm children, stoke division, weaken our democracy, and much more” – former Facebook employee
  - Leaked tens of thousands of pages of internal documents
  - Hope for government regulation

- Internal research found toxic risks of Instagram
  - Threat to teen girls’ mental health
  - Prevalence of drug cartels and human traffickers

- Can follow WSJ investigative series

More reading that may be of interest!

- Are drones suitable tools for delivering medical supplies in developing countries? — GSMA
- This Aerial Photographer Captured Images of the Mass Burials on Hart Island. Then, the New York Police Department Confiscated His Drone — artnet
- Police Drones — New York Times
- FBI releases secret spy plane footage — ACLU
- Drone War — The Bureau Investigates
- Whistleblower to Congress: Facebook products harm children and weaken democracy — NPR
- The Facebook Files — WSJ