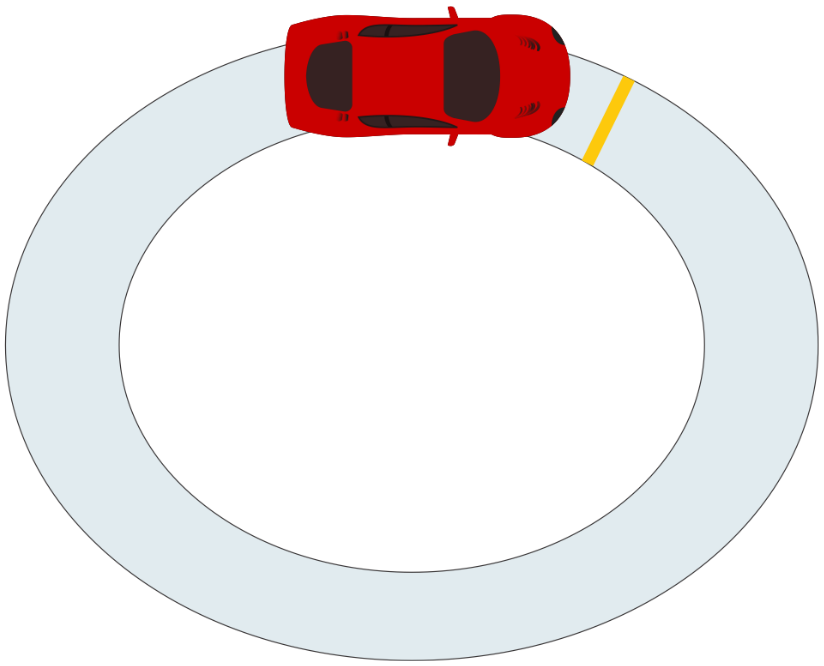
Set Cover Problem

for loops, object references, copying objects and arrays of objects, sharing data

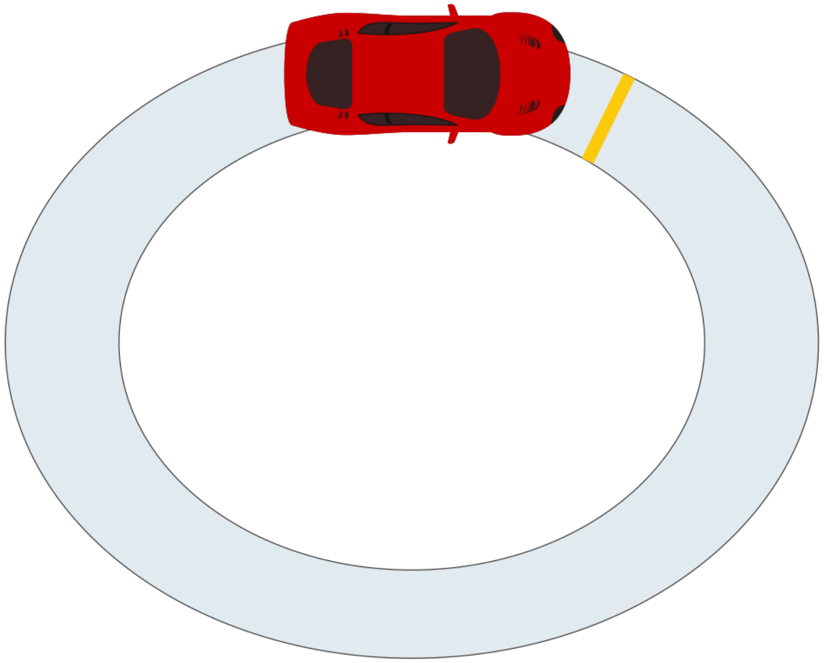
for loop

A new kind of loop for when we know exactly how many times to drive around the track.



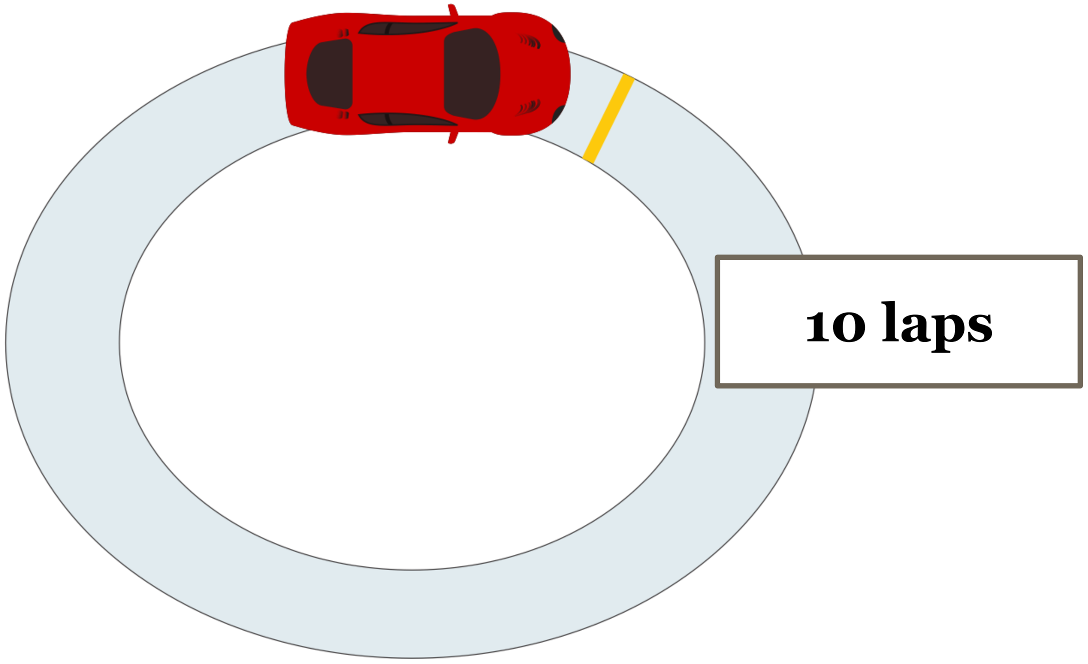
Drive the same track multiple times

while loop



Drive the track while the race is not over

for loop



Drive the track exactly ten times.

for loop

```
for (int counter=0; counter < 10; counter++)
{
    println(counter);
}
```

for loop

declare and assign a loop variable

```
for (int counter=0; counter < 10; counter++)
{
     println(counter);
}
```

for loop

check whether loop should end

```
for (int counter=0; counter < 10; counter++)
{
     println(counter);
}
```

for loop

```
for (int counter=0; counter < 10; counter++)
{
    println(counter);
}
```

run the body

for loop

```
for (int counter=0; counter < 10; counter++)
{
    println(counter);
}
```

adjust the loop variable

for versus while

```
for (int counter=0; counter < 10; counter++)
{
    println(counter);
}
```

...is equivalent to...

```
int counter = 0;
while (counter < 10)
{
    println(counter);
    counter++;
}
```

for versus while

Use `while` when you don't know how many iterations there will be

Use `for` when you know exactly how many iterations there will be

Examples of when while is better

Use while when you don't know how many iterations there will be

Examples of when while is better

Use while when you don't know how many iterations there will be

Use while when it's easier to come up with a stopping condition…

```
float radius = max(corners);
int colorIndex = startIndex;
while (radius > 0)
{
  fill(colors[colorIndex]);
  colorIndex = (colorIndex + 1);
  if (colorIndex >= colors.length)
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

```
float radius = max(corne
int colorIndex = startI
while (radius > 0)
{
  fill(colors[colorInde
  colorIndex = (colorInc
  if (colorIndex >= colc
  {
    colorIndex = 0;
  }

  ellipse(x, y, 2*radius, 2*radius);
  radius -= radiusChange;
}
```

We could calculate how many iterations it takes to get to radius zero, but this is much easier

```
while (left <= right)
{
  numIterations++;
  int mid = (left + ((right - left) / 2));
  if (numbers[mid] < valueToFind)
  {
    left = mid+1;
  }
  else if (numbers[mid] > valueToFind)
  {
    right = mid - 1;
  }
  else
  {
    foundIndex = mid;
    break;
  }
}
```

```
while (left <= right)
{
  numIterations++;
  int mid = (left + ((ri
  if (numbers[mid] < val
  {
    left = mid+1;
  }
  else if (numbers[mid] > valueToFind)
  {
    right = mid - 1;
  }
  else
  {
    foundIndex = mid;
    break;
  }
}
```

Easier to set up a stopping condition than count the number of iterations

Examples of when for is better

Use for when you know exactly how many iterations there will be

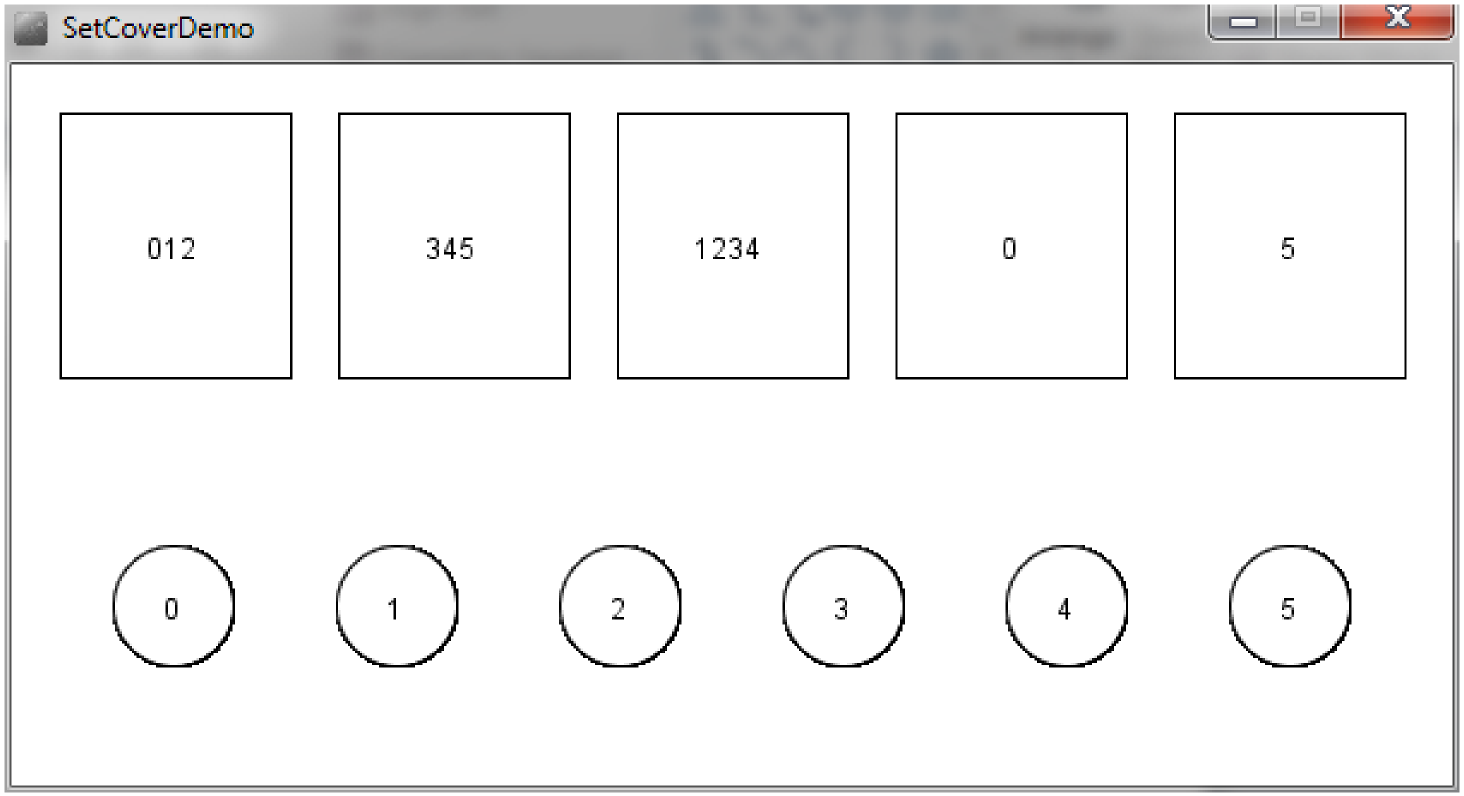
?

Examples of when for is better

Use for when you know exactly how many iterations there will be

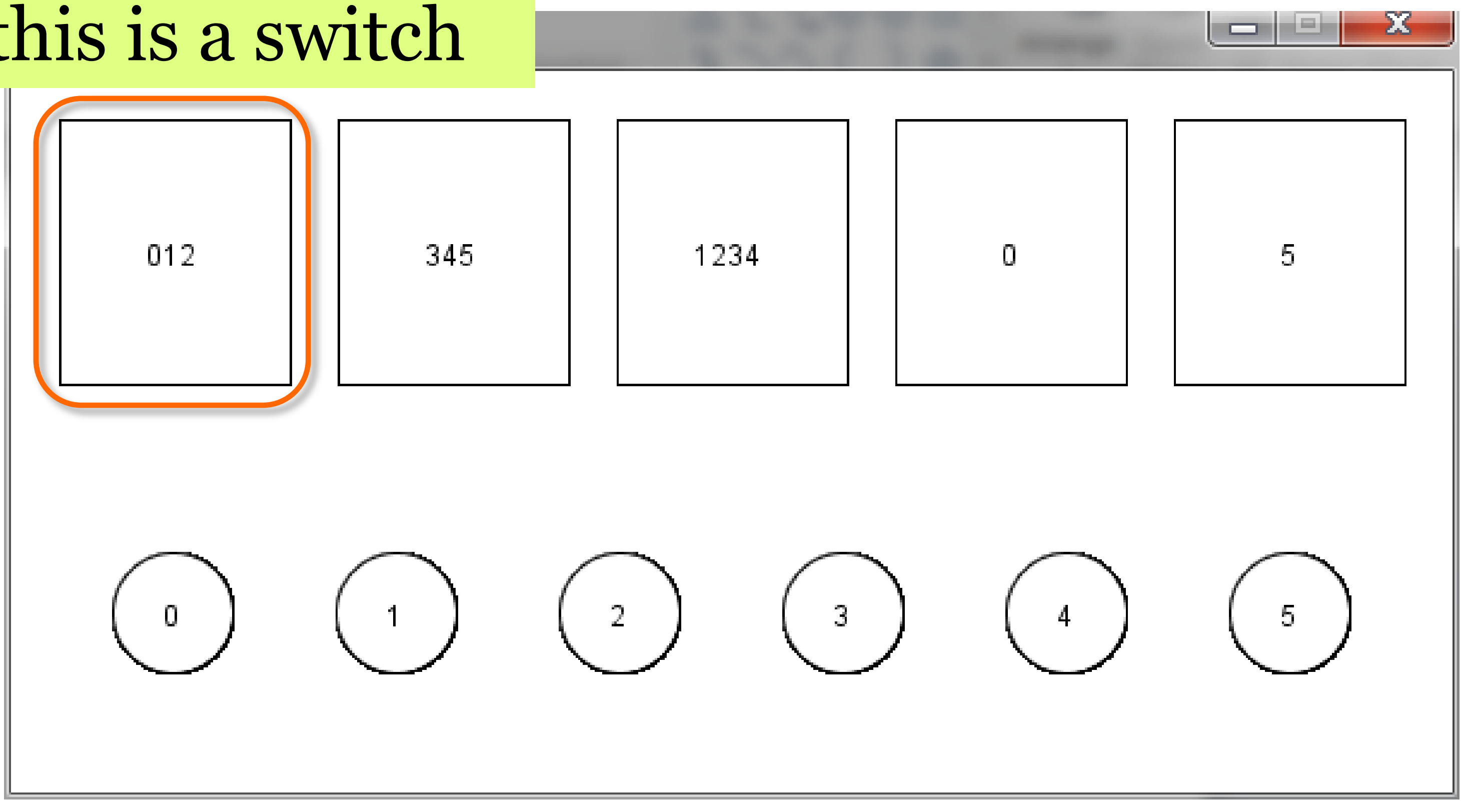
Examples:
Searching an array
Drawing items in an array
Doing something within a range of numbers

Set Cover Problem

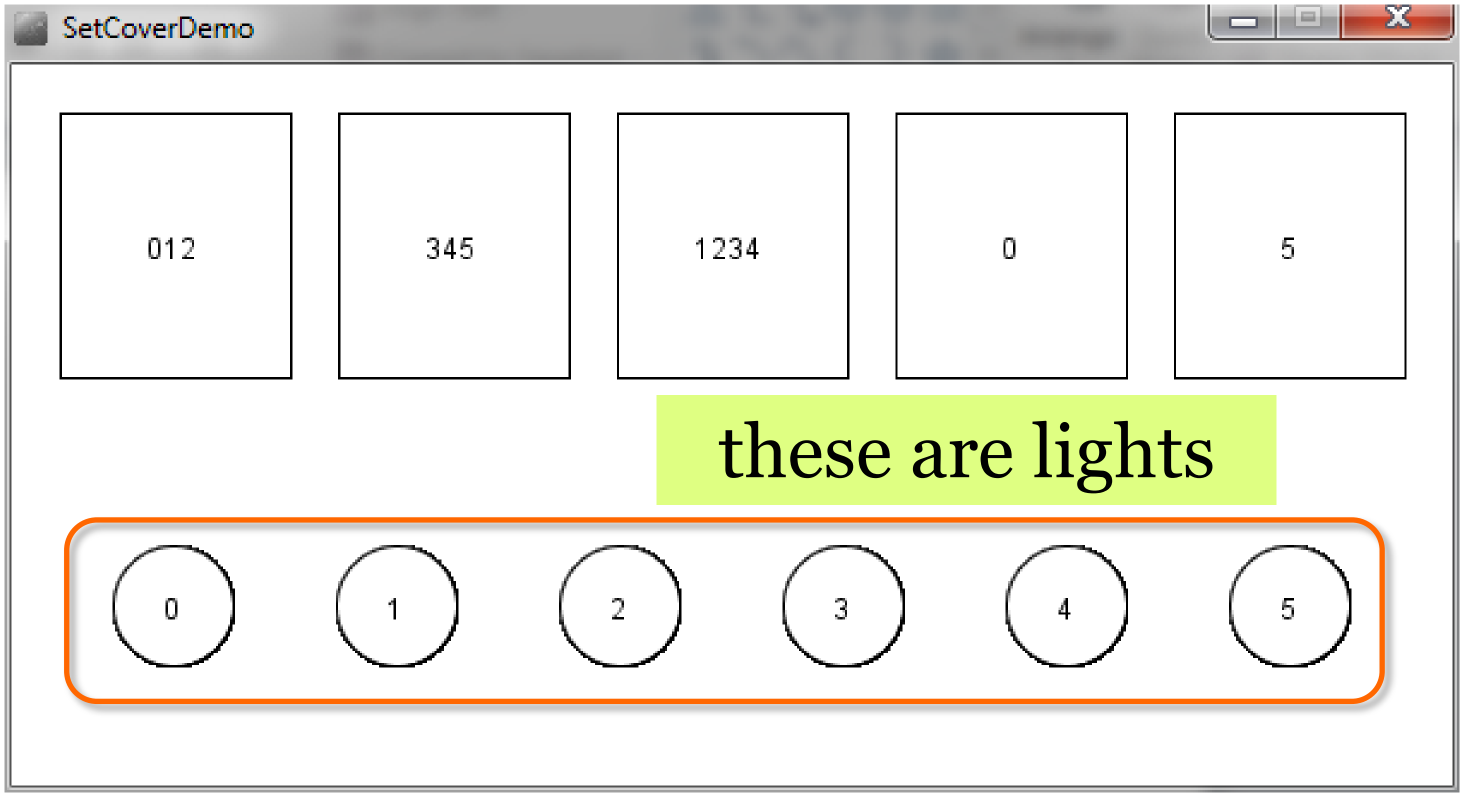


Set Cover Problem

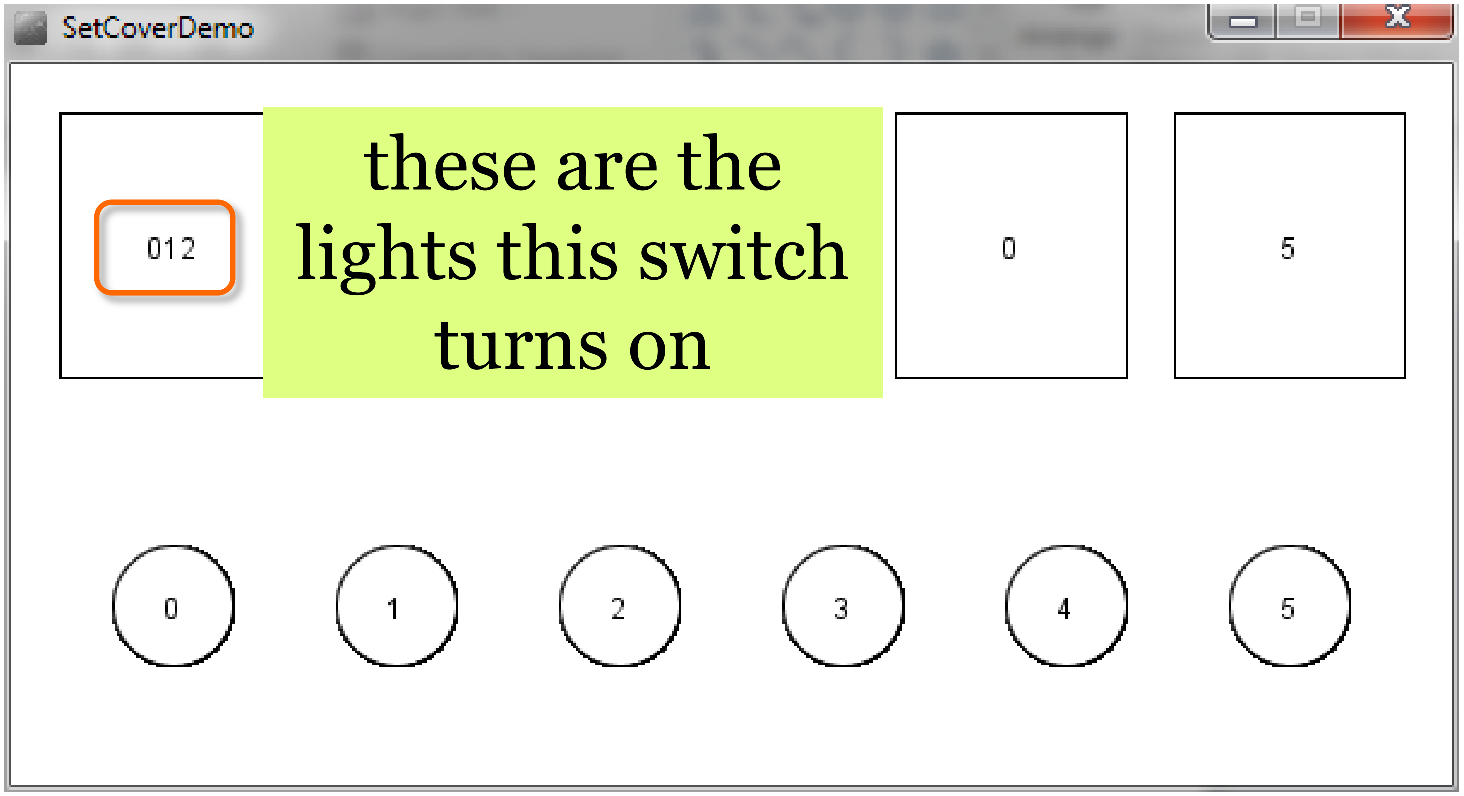
this is a switch



Set Cover Problem

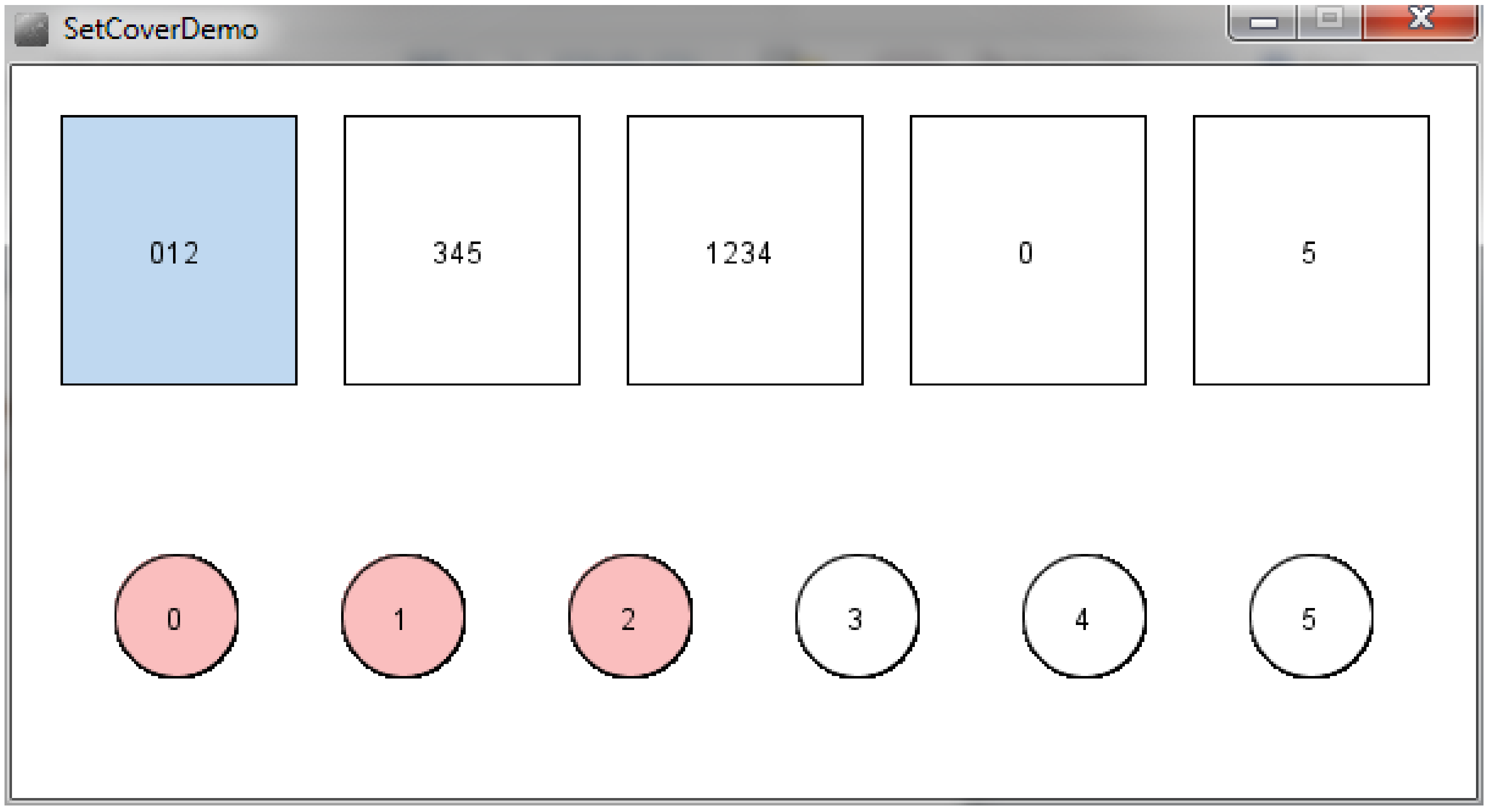


Set Cover Problem



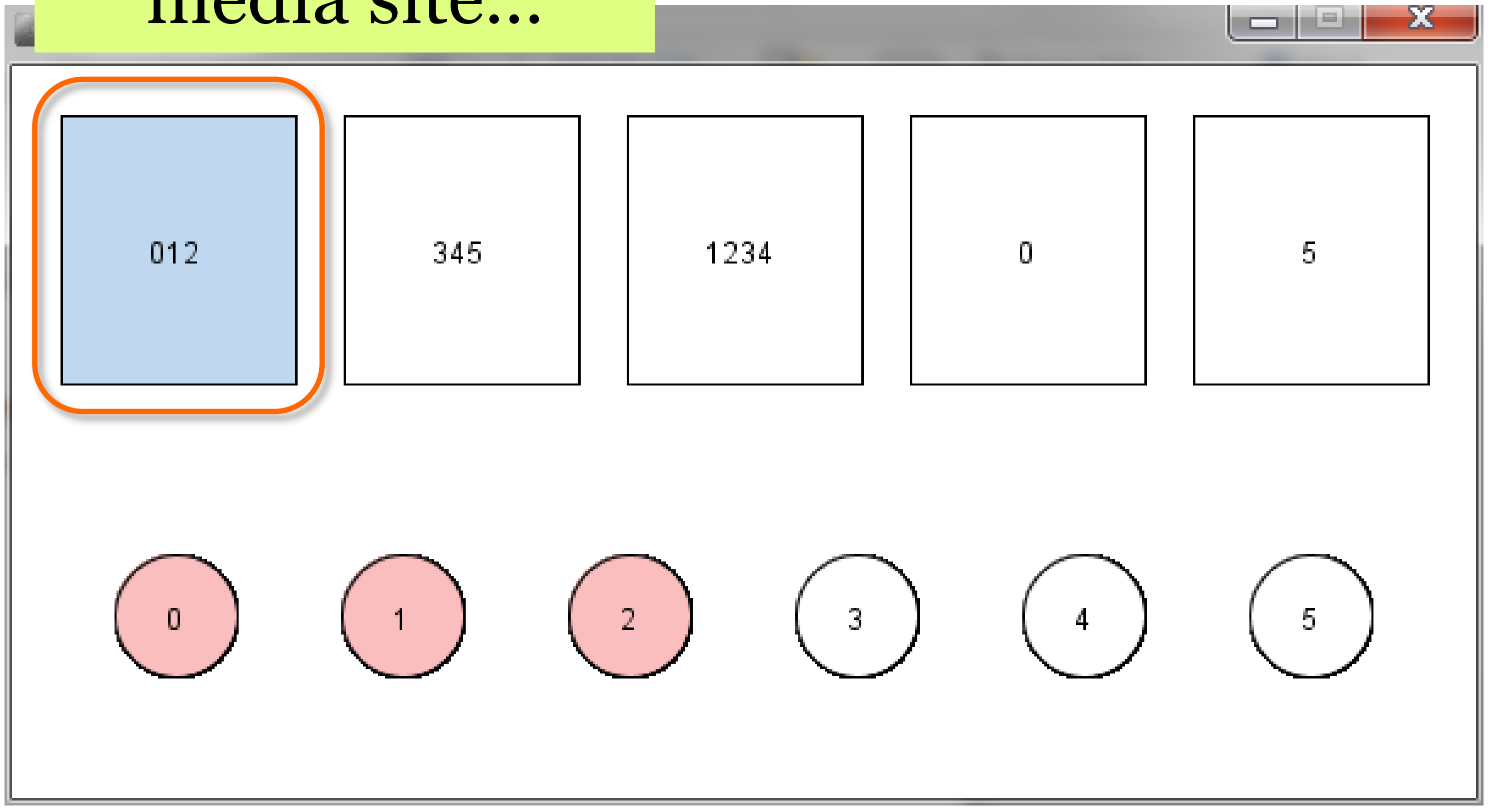
these are the lights this switch turns on

Set Cover Problem

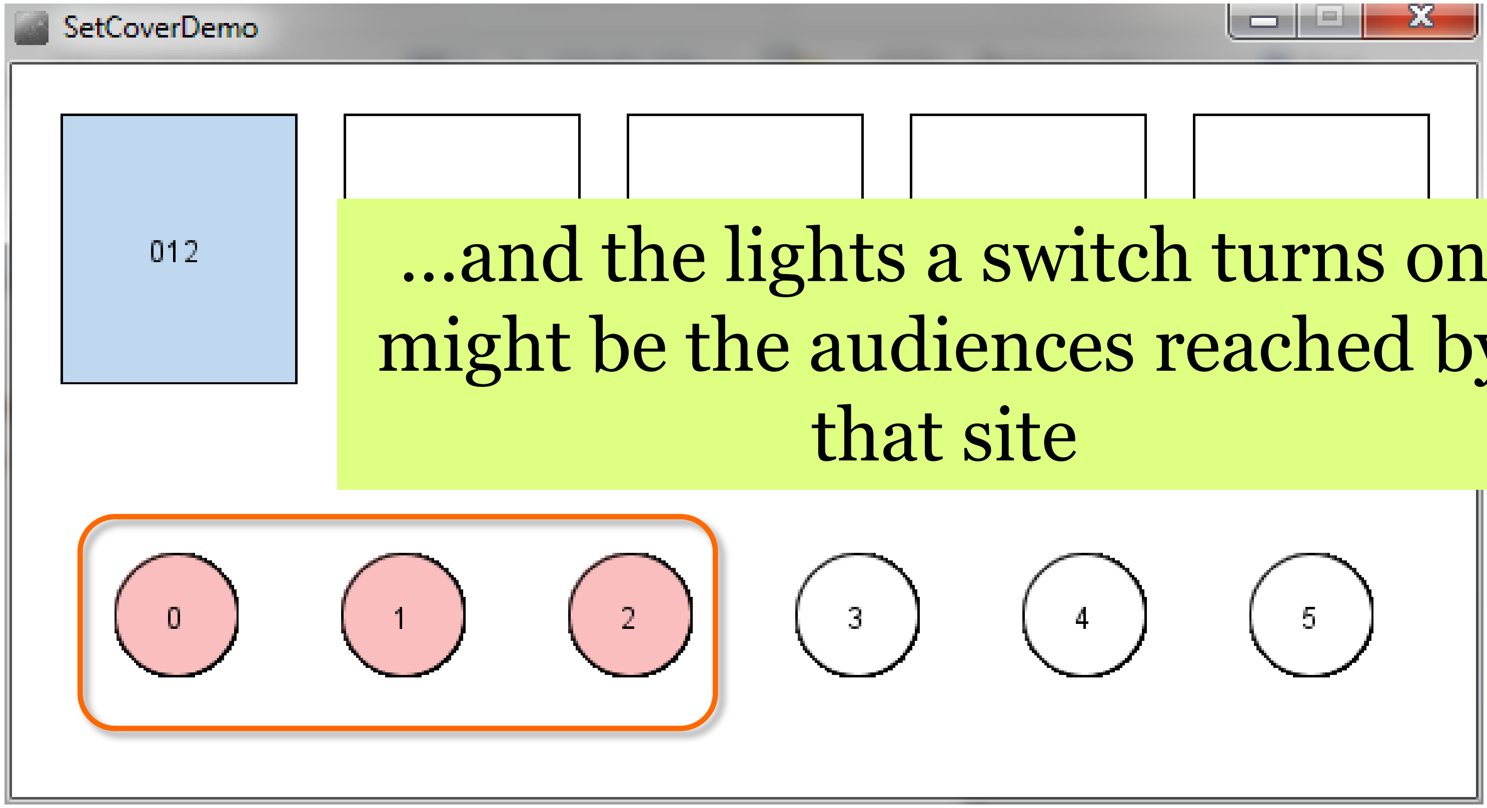


ver Problem

each switch might be a social media site…

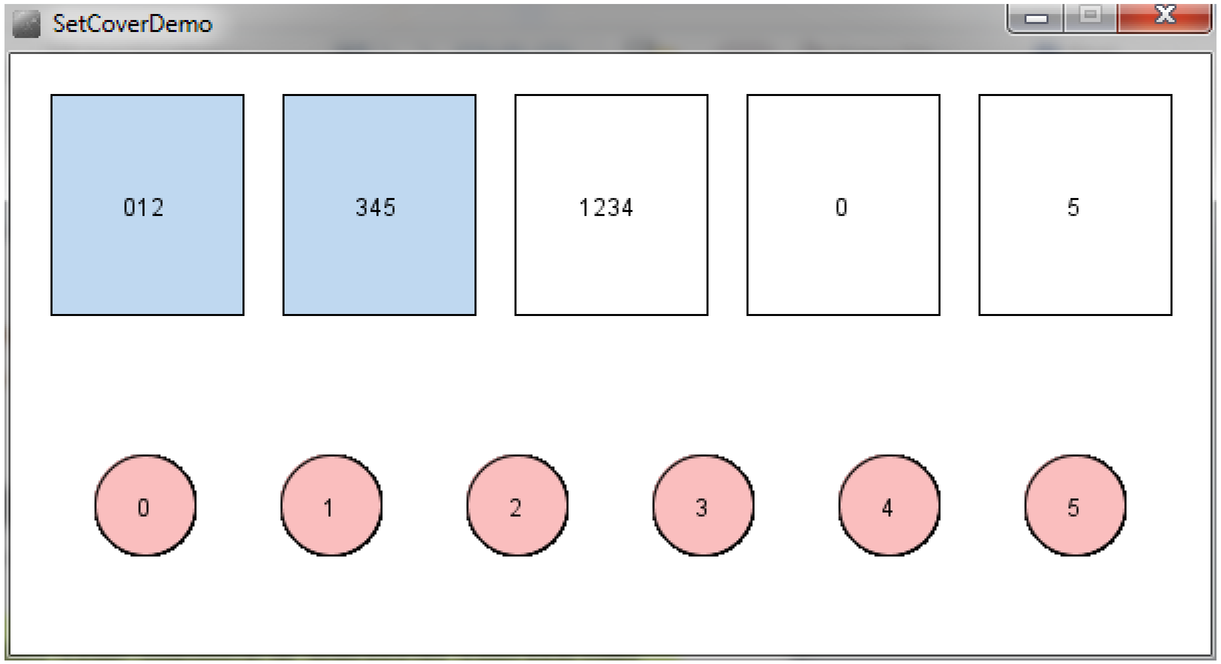


Set Cover Problem



...and the lights a switch turns on might be the audiences reached by that site

Set Cover Problem



Set cover problem: What switches should we press to turn on all the lights so the number of switches is minimized?

class Light

Attributes:
int x
int y
int num
int size
boolean on

class Light

Attributes:

int x
int y
int num
int size
boolean on

location on the screen

class Light

Attributes:

int x

int y

int num

int size

boolean on

like a unique ID used to label lights; can also be used to index an array

class Light

Attributes:

int x

int y

int num

$\boxed{\text{int size}}$

boolean on

set automatically

class Light

Attributes:
int x
int y
int num
int size
boolean on

true if light is on because of a switch

class Light

Attributes:
int x
int y
int num
int size
boolean on

class Switch

Attributes:
int x
int y
int width
int height
boolean on
Light[] lightsTurnedOn

class Light

Attributes:
int x
i
int
int size
boolean on

similar to Light's attributes

class Switch

Attributes:
int x
int y
int width
int height
boolean on
Light[] lightsTurnedOn

class Light

Attributes:
int x
int y
int num
int size
boolean on

class Switch

Attributes:
int x
int y
int width
int height
boolean on
Light[] lightsTurnedOn

an array of Light objects this switch turns on

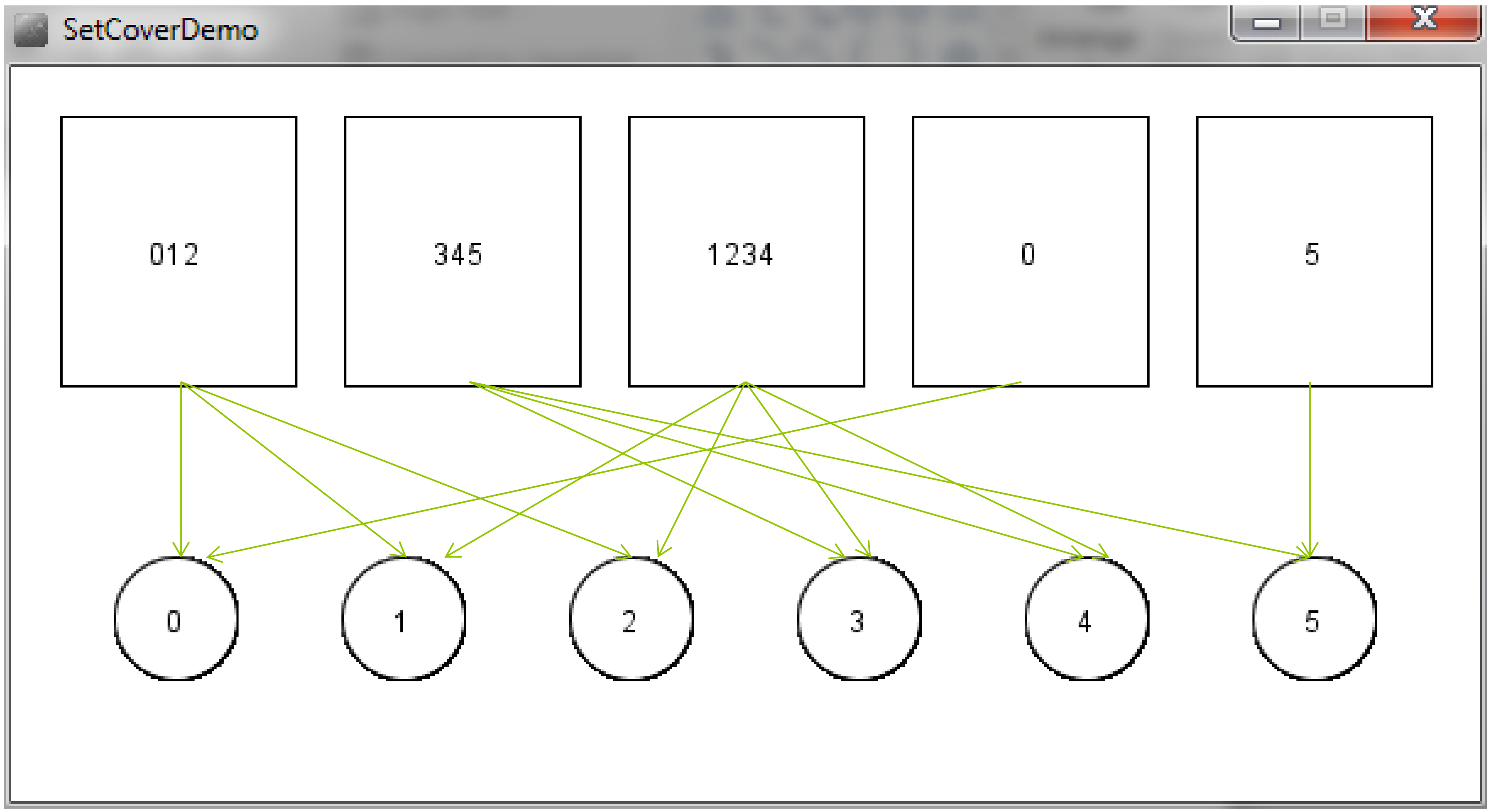
class Light

Attributes:
int x
int y
int num
int size
boolean on

class Switch

Attributes:
int x
int y
int width
int height
boolean on
Light[] lightsTurnedOn

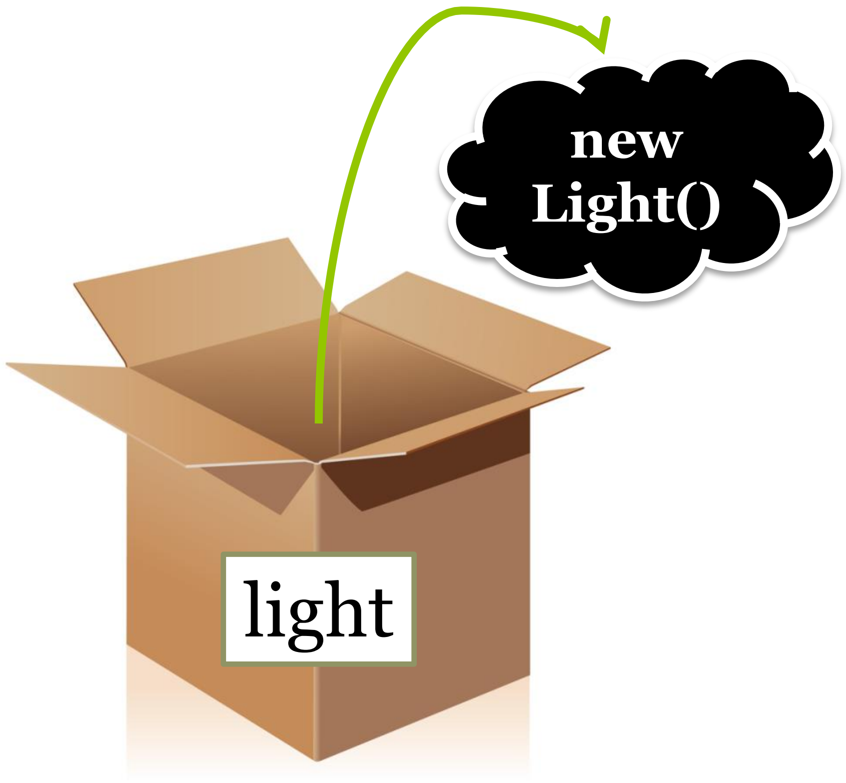
Should each Switch have its own copies of Lights?


SetCoverDemo
012
345
1234
0
5
0
1
2
3
4
5

Primitives vs. Objects

200
catHeadX

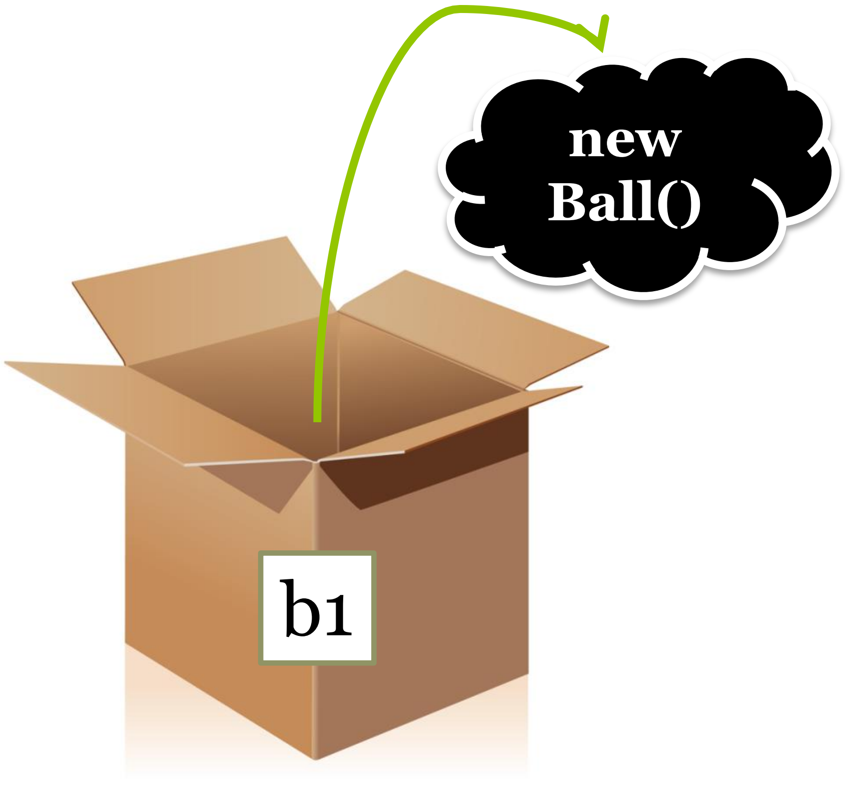
```
int catHeadX = 200;
```



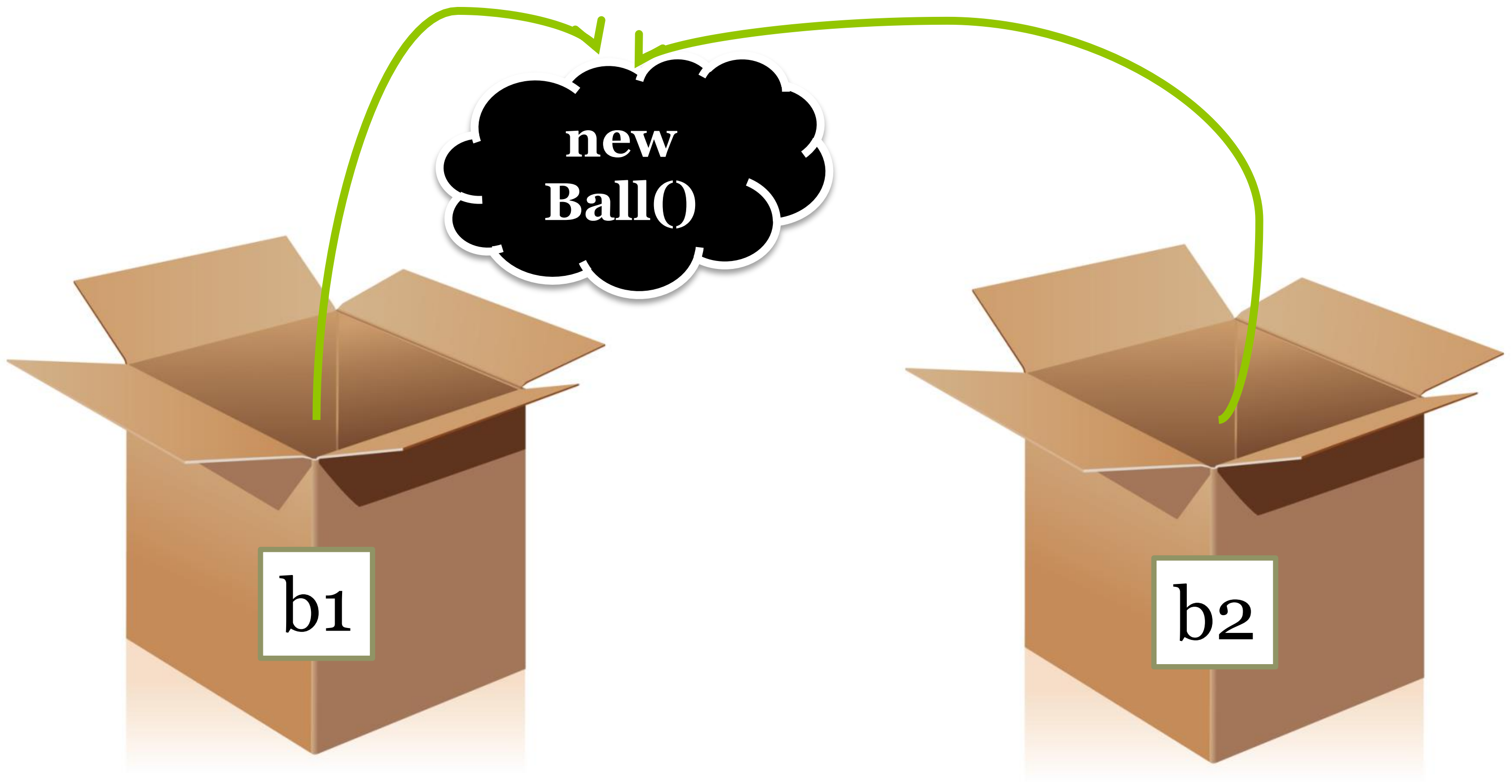
```
Light light = new
      Light();
```

```
class Ball
{
  int x;
  int y;
}

Ball b1 = new Ball();
b1.x = 10;
b1.y = 20;

Ball b2 = b1;

b1.x = 25;
b1.y = 45;

println(b2.x);
```

What is the output of this code?



```
Ball b1 = new Ball();
```



```
Ball b1 = new Ball();
```

```
Ball b2 = b1;
```

Poll Everywhere Question

```
class Ball
{
  int x;
  int y;
  Ball(int newx, int newy)
  {
    x = newx;
    y = newy;
  }
}

class BallPit
{
  Ball[] balls;
}
```

```
BallPit pit = new BallPit();
pit.balls = new Ball[2];
pit.balls[0] = new Ball(10, 20);
pit.balls[1] = new Ball(45, 55);

// Copy the ball pit
BallPit pit2 = new BallPit();
pit2.balls = pit.balls;

pit2.balls[0].x = 100;

println(pit.balls[0].x);
```

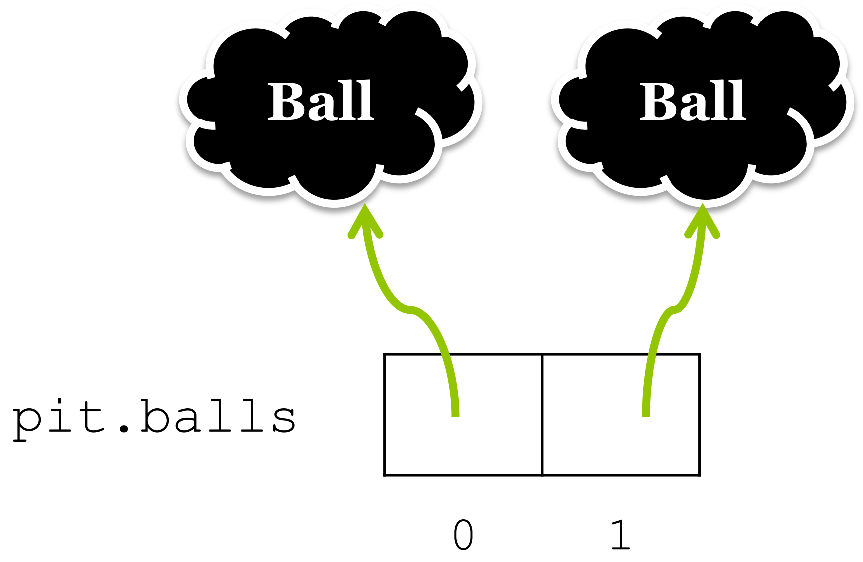
What will be printed?
Text 37607
444946: 10
444947: 45
444948: 100
444949: nothing/error?

Copying Objects

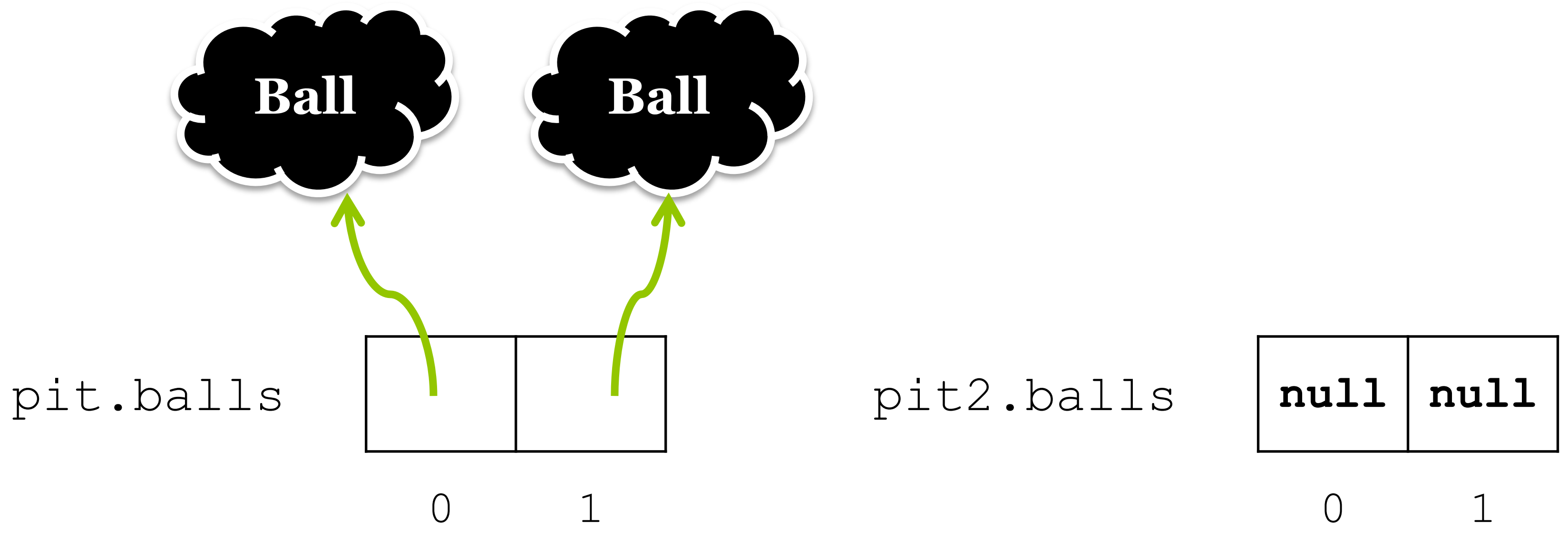
Shallow Copy:
Copy just the variables inside the object.

Deep Copy:
Every time a variable stores a reference, repeat the full copy process on the object stored.

Copying Arrays of Objects (Shallow)

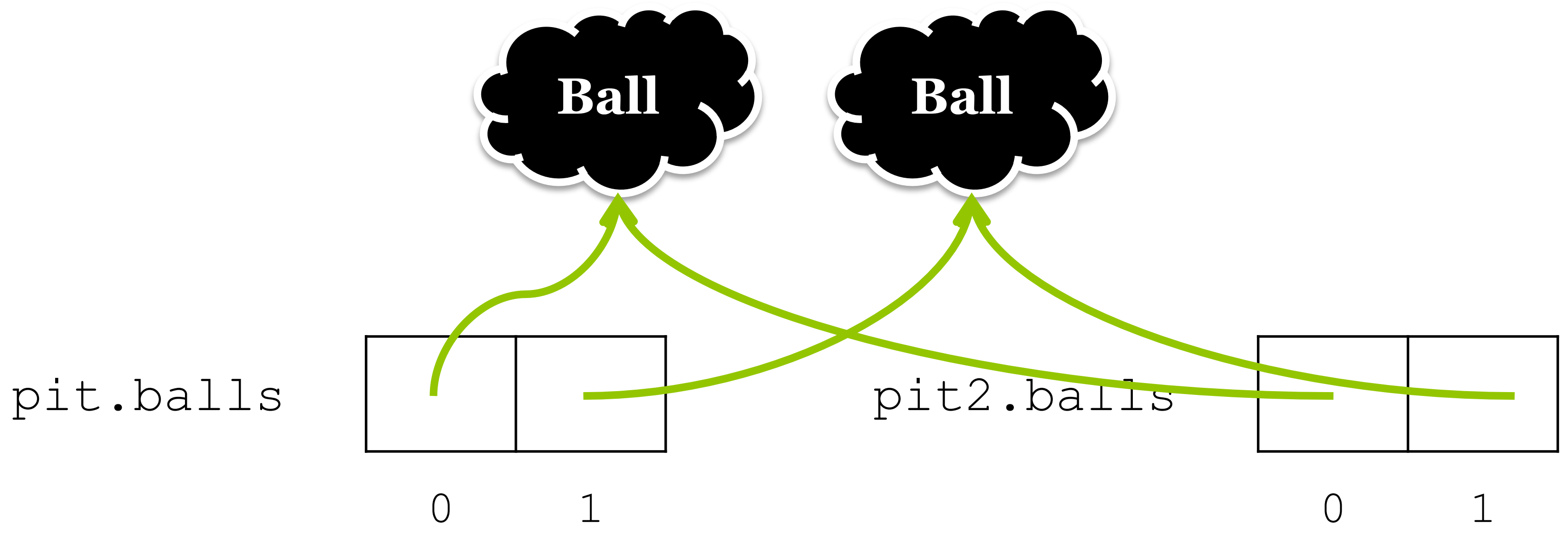


Creating a New Array of Objects



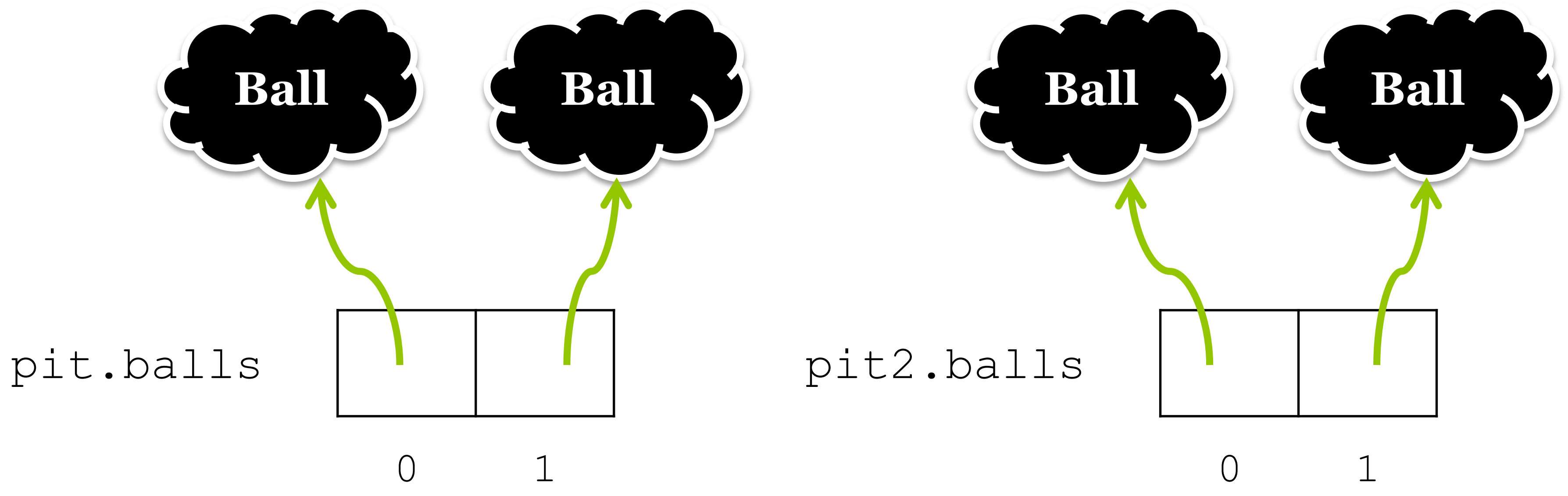
```
BallPit pit2 = new BallPit();
pit2.balls = new Ball[pit.balls.length];
```

Copying Arrays of Objects (Shallow)

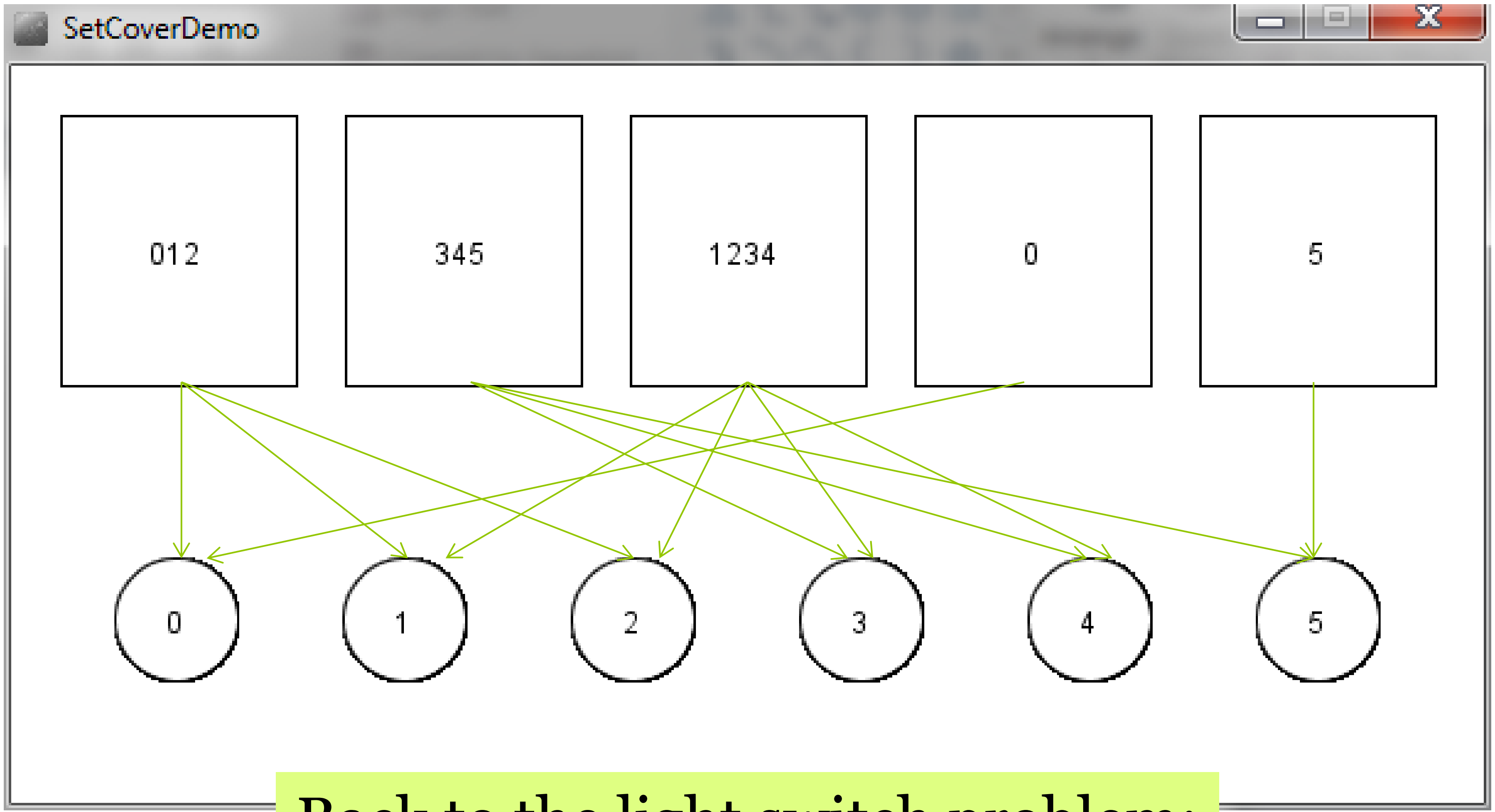


```
for (int ballNum=0; ballNum < pit.balls.length; ballNum++)
{
    pit2.balls[ballNum] = pit.balls[ballNum];
}
```

Copying Arrays of Objects (Deep)



```
for (int ballNum=0; ballNum < pit.balls.length; ballNum++)
{
   pit2.balls[ballNum] = new Ball(pit.balls[BallNum].x,
                                  pit.balls[BallNum].y);
}
```



Back to the light switch problem: each `Switch` only needs to store references to `Lights` – not individual copies of them.

```
class Light
{
  int x;
  int y;
  int num;

  int size;

  boolean on;

  Light(int newX, int newY, int newNum)
  {
    x = newX;
    y = newY;
    num = newNum;

    size = lightSize;
    on = false;
  }
}
```

```
class Switch
{
  int x;
  int y;
  int width;
  int height;

  boolean on;

  Light[] lightsTurnedOn;

  Switch(int newX, int newY, int newWidth, int newHeight)
  {
    x = newX;
    y = newY;
    width = newWidth;
    height = newHeight;

    lightsTurnedOn = new Light[numLights];
    on = false;
  }
}
```

```
class Switch
{
  int x;
  int y;
  int width;
  int height;

  boolean on;

  Light[] lightsTurnedOn;

  Switch(int newX, int newY                                eight)
  {
    x = newX;
    y = newY;
    width = newWidth;
    height = newHeight;

    lightsTurnedOn = new Light[numLights];
    on = false;
  }
}
```

array of Lights is really an array of references to Lights

```
lights = new Light[numLights];
for (int lightNum=0; lightNum < lights.length; lightNum++)
{
  float x = spaceBetweenLights * (lightNum+1) +
            lightSize*lightNum + lightSize/2;

  lights[lightNum] = new Light((int)x, height*3/4,
                               lightNum);
}
```

create some lights first (one for each number between 0 and 5 inclusive)

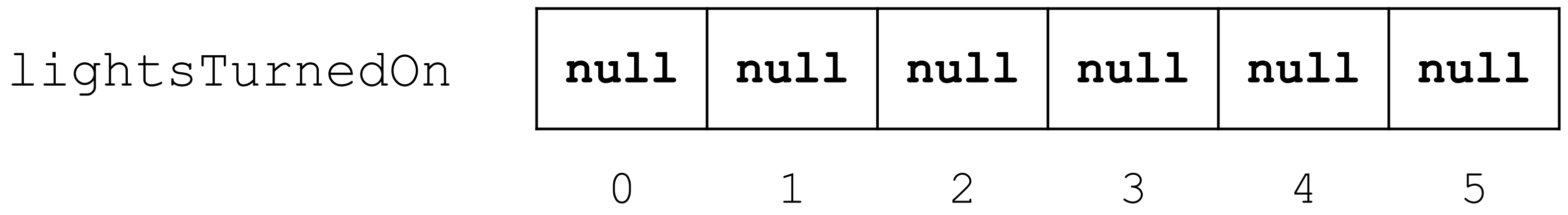
```
switches = new Switch[numSwitches];
for (int switchNum=0; switchNum < switches.length;
switchNum++)
{
  float x = spaceBetweenSwitches * (switchNum+1) +
            switchSize*switchNum;

  switches[switchNum] =
    new Switch((int)x, 20,(int)switchSize, height/2 - 40);
}
```

create some switches (the array of Lights will be created by the Switch constructor)

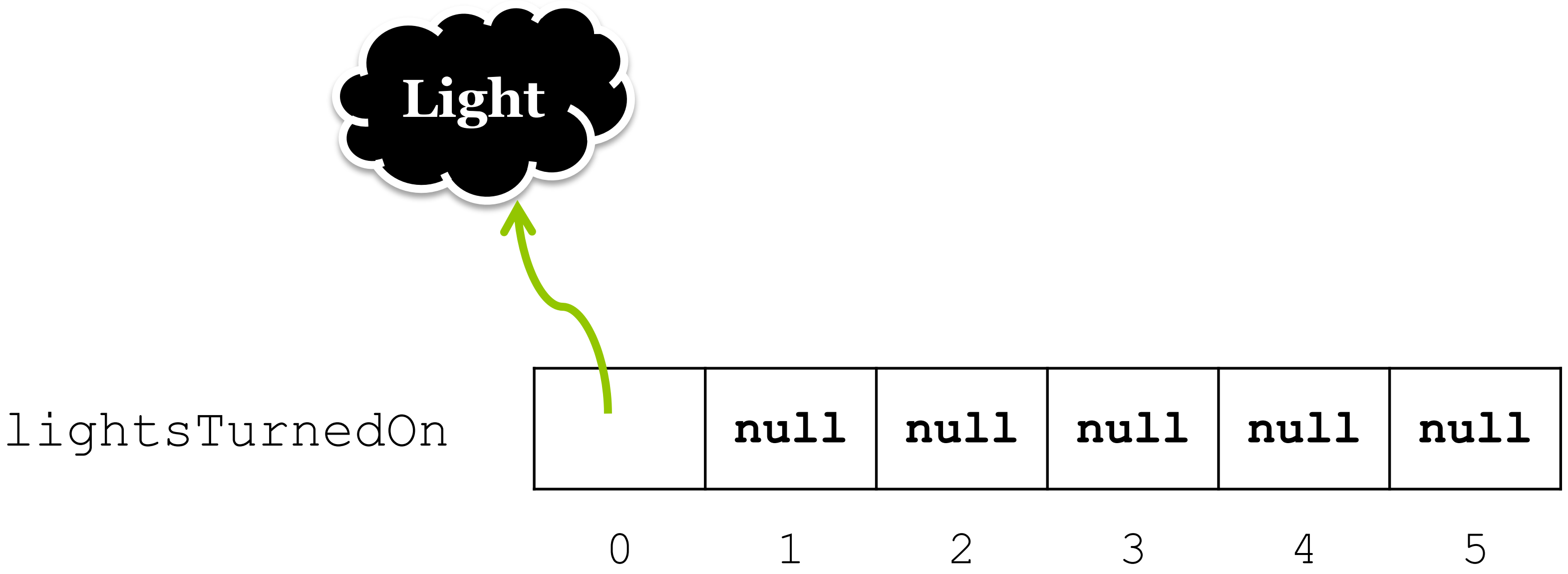
```
switches[0].lightsTurnedOn[lights[0].num] = lights[0];
switches[0].lightsTurnedOn[lights[1].num] = lights[1];
switches[0].lightsTurnedOn[lights[2].num] = lights[2];

switches[1].lightsTurnedOn[lights[3].num] = lights[3];
switches[1].lightsTurnedOn[lights[4].num] = lights[4];
switches[1].lightsTurnedOn[lights[5].num] = lights[5];

switches[2].lightsTurnedOn[lights[1].num] = lights[1];
switches[2].lightsTurnedOn[lights[2].num] = lights[2];
switches[2].lightsTurnedOn[lights[3].num] = lights[3];
switches[2].lightsTurnedOn[lights[4].num] = lights[4];

switches[3].lightsTurnedOn[lights[0].num] = lights[0];

switches[4].lightsTurnedOn[lights[5].num] = lights[5];
```

Finally add the `Lights` to the `Switch` arrays (references will be stored, not copies of the objects)

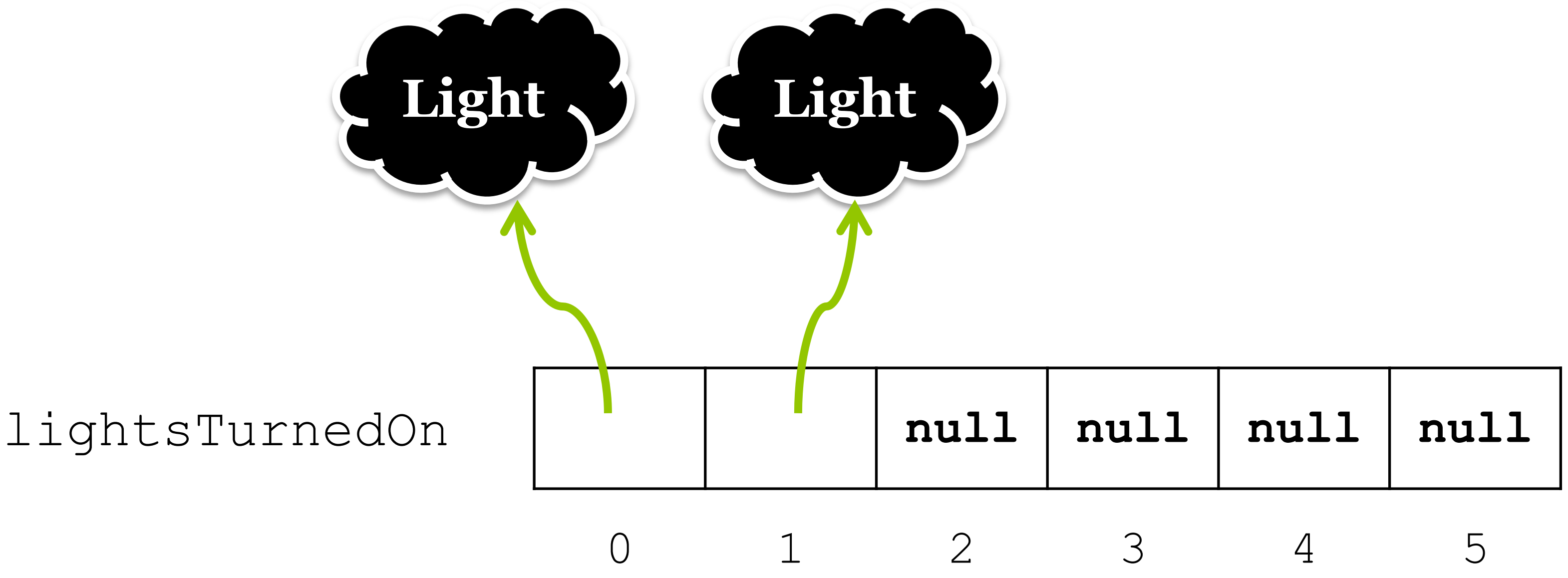
What happens when we don't use the whole `lightsTurnedOn` array?



```
lightsTurnedOn = new Light[numLights];
```



```
lightsTurnedOn[0] = lights[0];
```



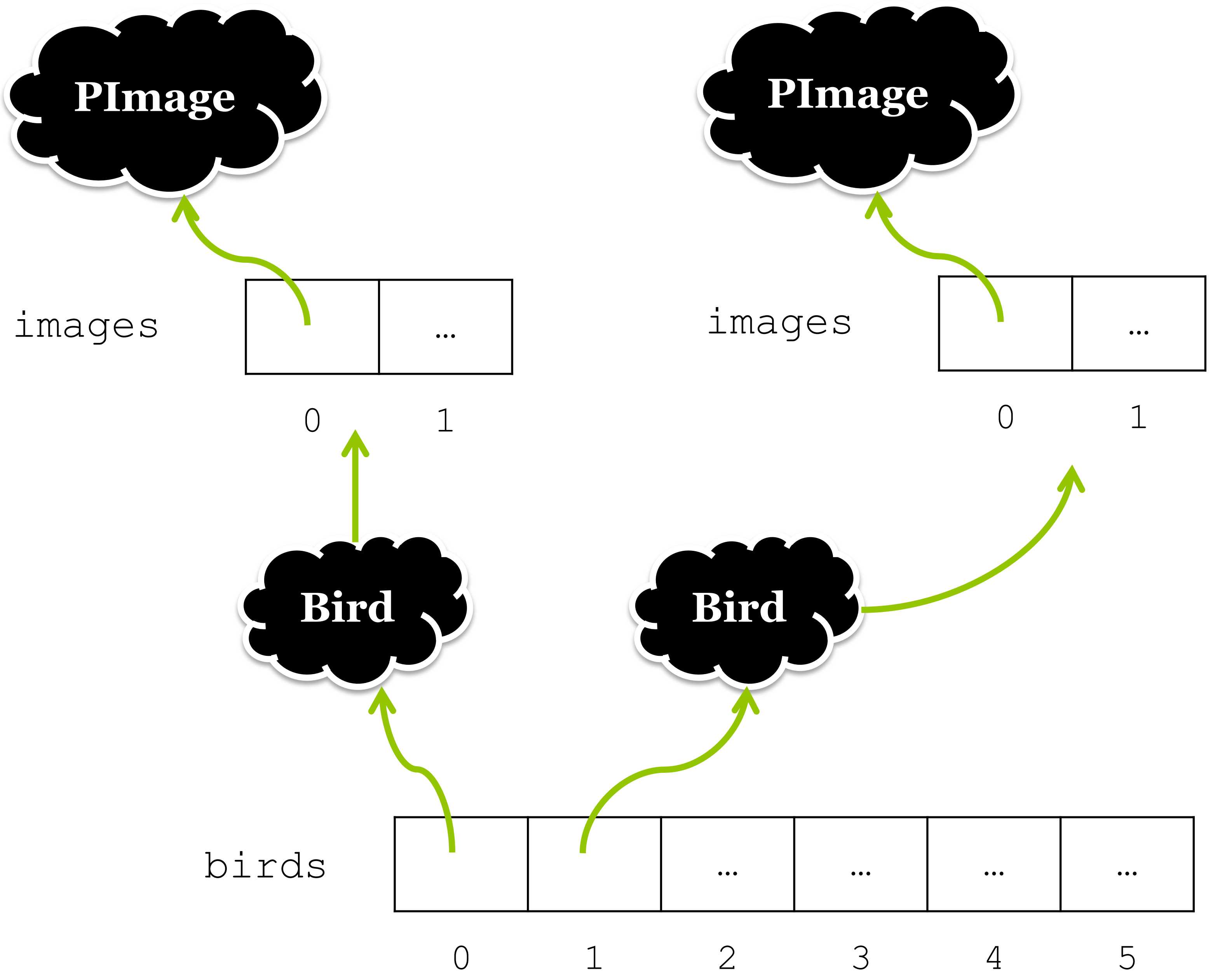
```
lightsTurnedOn[1] = lights[1];
```

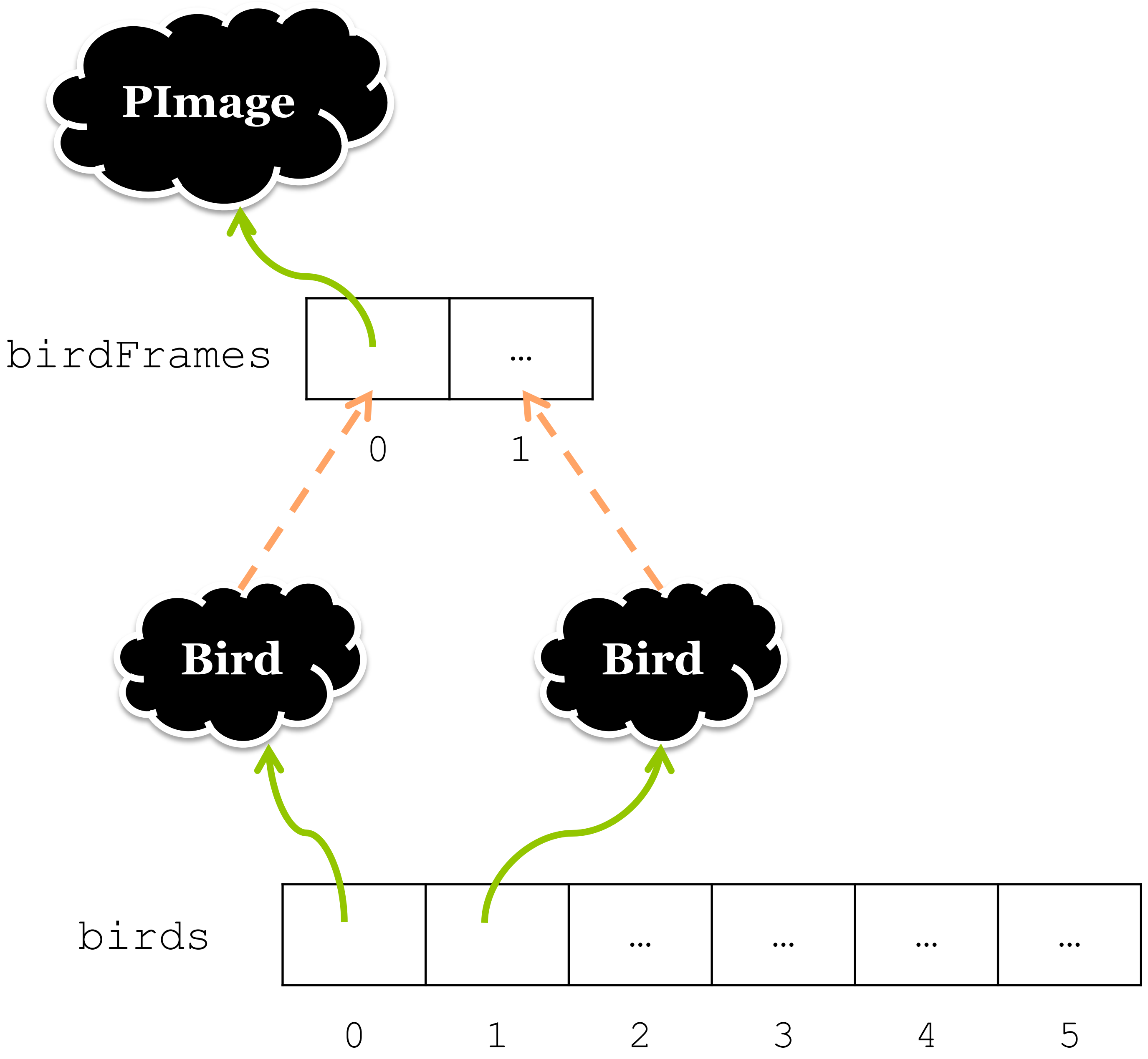
```
String numList = ""; // <- empty String
for (int lightNum=0;
     lightNum < aSwitch.lightsTurnedOn.length;
     lightNum++)
{
  if (aSwitch.lightsTurnedOn[lightNum] != null)
  {
    numList += aSwitch.lightsTurnedOn[lightNum].num;
  }
}
```

```
String numList = ""; // <- empty String
for (int lightNum=0;
     lightNum < aSwitch.lightsTurnedOn.length;
     lightNum++)
{
  if (aSwitch.lightsTurnedOn[lightNum] != null)
  {
    numList += aSwitch.lightsTurnedOn[lightNum].num;
  }
}
```

when going through the array, check for null before doing something with the object

When Sharing Data Matters


PImage
PImage
images
...
0
1
images
...
0
1
Bird
Bird
birds
...
...
...
...
0
1
2
3
4
5


PImage
birdFrames
...
0
1
Bird
Bird
birds
...
...
...
...
0
1
2
3
4
5