COMP 1406: Introduction to Objects

Objects Review

Objects in C++

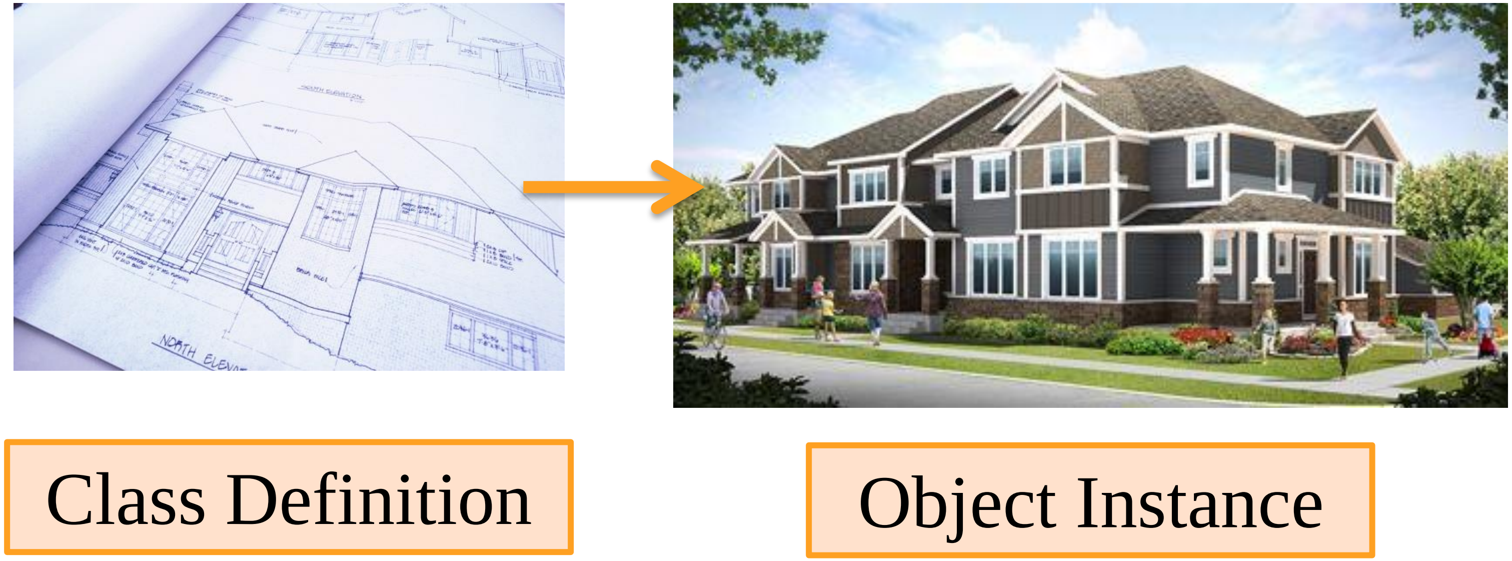
Objects in Java

Objects Review

Classes

Objects

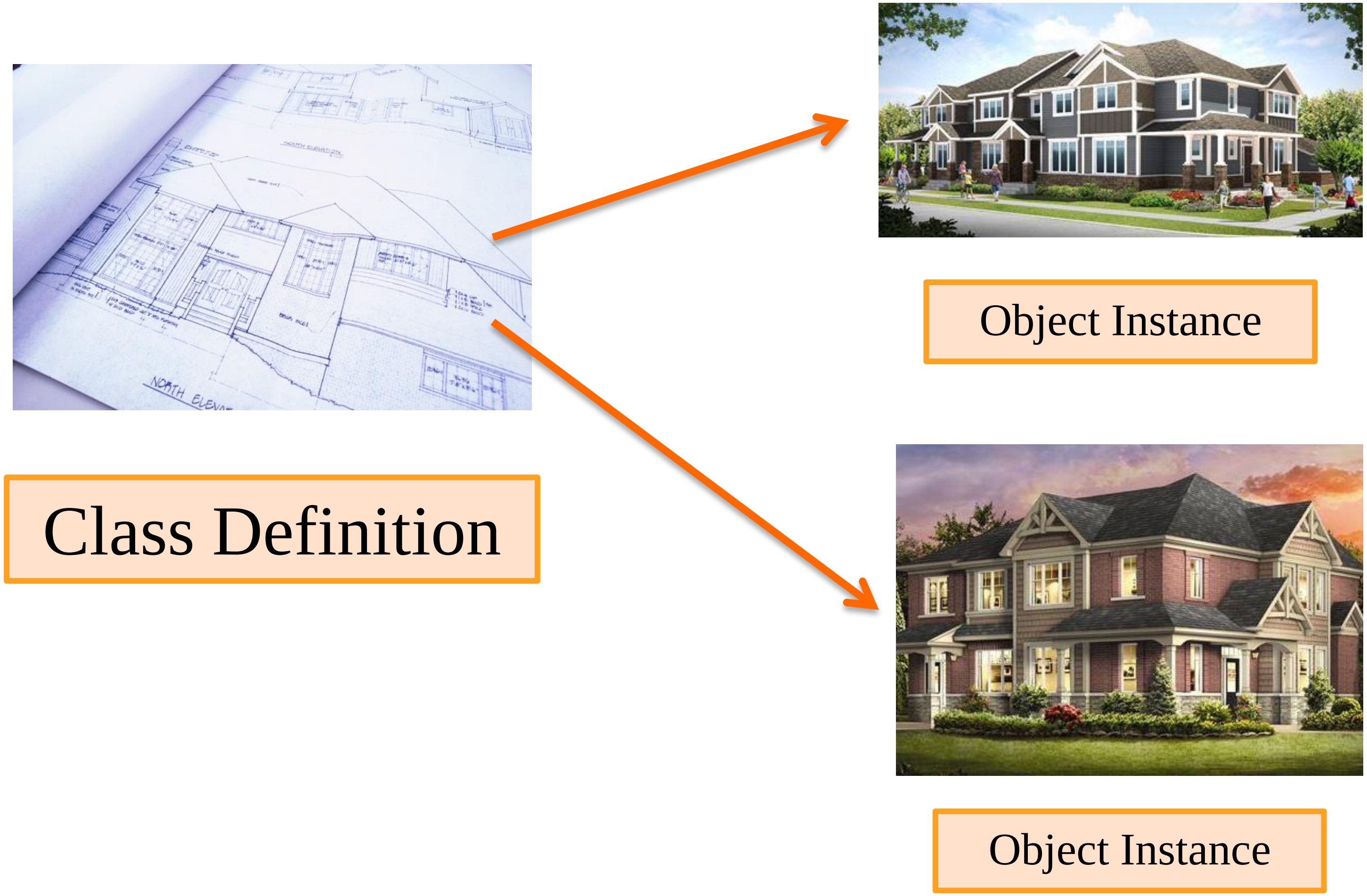
Objects



Class Definition

Object Instance

Objects



Class Definition

Object Instance

Object Instance

Objects

NORTH ELEVA

Class Definition

Variables
Methods/Member functions

Object Instance

Variables
Methods/Member functions

Defining a Class

In Processing:

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

Ball b = new ball();
b.x = 10;
b.y = 20;
```

In Python:

```
class Ball:

   def __init__(self):
      pass

b = Ball()
b.x = 10
b.y = 20
```

Objects in C++

Defining a Class in C++

```
class ball
{
public:
  int x;
  int y;
};
```

Defining a class in C++ is almost the same as defining a struct.

Defining a Class in C++

```
class ball
{
public:
   int x;
   int y;
};
```

The main difference from a struct is that we use the `public` keyword. We'll see what it means soon…

Creating an Object in C++

```
class ball
{
public:
  int x;
  int y;
};

ball b;
b.x = 10;
b.y = 20;
```

We can declare a variable of our class type, just like a struct…

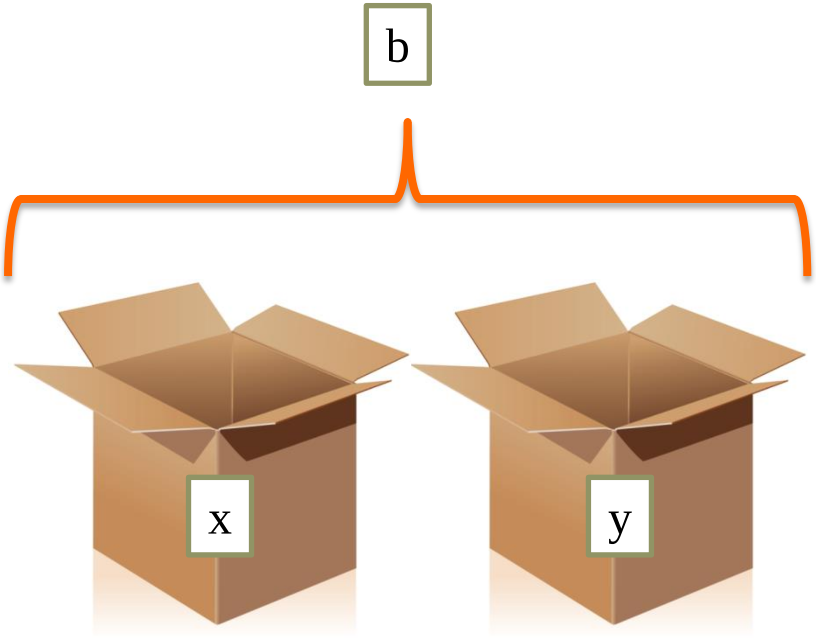
Creating an Object in C++

```
class ball
{
public:
   int x;
   int y;
};

ball b;
b.x = 10;
b.y = 20;
```

…then start assigning data to the variables in memory.

Creating an Object in C++

```
class ball
{
public:
  int x;
  int y;
};

ball b;
b.x = 10;
b.y = 20;
```



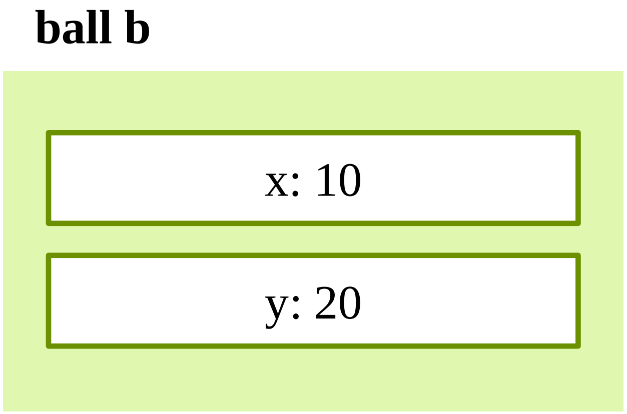
Creating an Object in C++

```
class ball
{
public:
   int x;
   int y;
};

ball b;
b.x = 10;
b.y = 20;
```

Memory Address	Identifier		Data Stored
500	b	x	10
501			
502			
503			
504		y	20
505			
506			
507			

Creating an Object in C++

```
class ball
{
public:
   int x;
   int y;
};

ball b;
b.x = 10;
b.y = 20;
```



A simplified view: imagine x and y as boxes stored contiguously in memory.

Pass By Reference With C++ Classes

Works the same way as structs – you have to use `&` if you don't want to pass by value (i.e. copy the data).

```
void moveBall(ball &b)
{
   b.x += 5;
}

void main()
{
   ball b;
   b.x = 10;
   b.y = 20;

   moveBall(b);

   return 0;
}
```

Dynamically Allocated Objects in C++

```
void moveBall(ball *b)
{
   b->x += 5;
}

void main()
{
   ball *b = new ball();
   b->x = 10;
   b->y = 20;

   moveBall(b);

   delete b;

   return 0;
}
```

Dynamically Allocated Objects in C++

```
void moveBall(ball *b)
{
    b->x += 5;
}

void main()
{
    ball *b = new ball();
    b->x = 10;
    b->y = 20;

    moveBall(b);

    delete b;

    return 0;
}
```

A `ball` is created on the heap and saved to a pointer `b`

Dynamically Allocated Objects in C++

```
void moveBall(ball *b)
{
    b->x += 5;
}

void main()
{
    ball *b = ne
    b->x = 10;
    b->y = 20;

    moveBall(b);

    delete b;

    return 0;
}
```

Recall that
`(*b).x`
is the same as
`b->x`

Dynamically Allocated Objects in C++

```
void moveBall(ball *b)
{
   b->x += 5;
}

void main()
{
   ball *b = new ball();
   b->x = 10;
   b->y = 20;

   moveBall(b);

   delete b;

   return 0;
}
```

Now we are passing a pointer value to our function

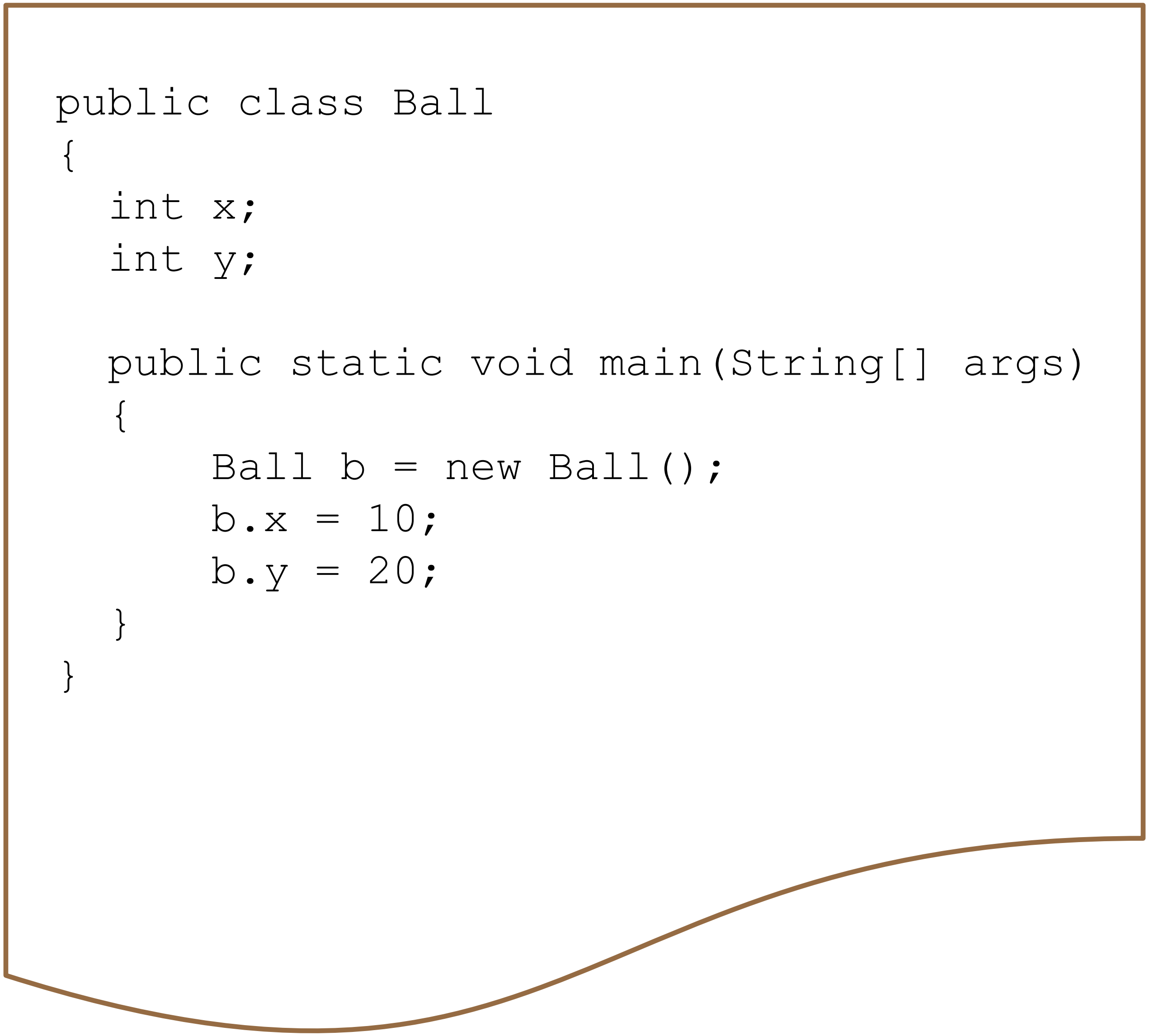
Dynamically Allocated Objects in C++

```
void moveBall(ball *b)
{
   b->x += 5;
}

void main()
{
   ball *b = new ball();
   b->x = 10;
   b->y = 20;

   moveBall(b)

   delete b;

   return 0;
}
```

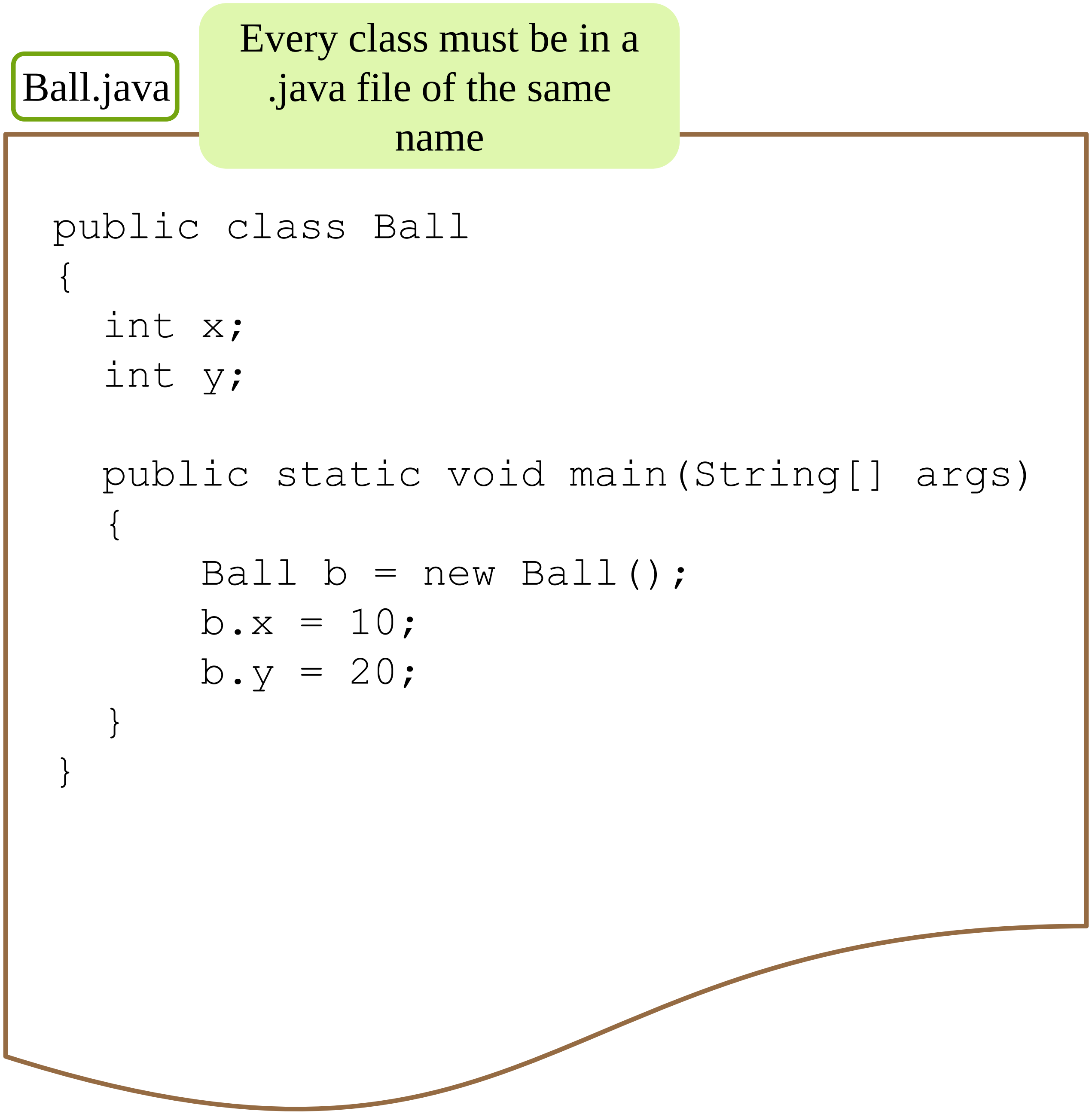
We must remember to clean up our memory

Objects in Java

Ball.java



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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```



Every class must be in a .java file of the same name

Ball.java

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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

Ball.java

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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

As in C++, this creates a new type called Ball, but there are no balls in memory yet.

Ball.java

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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

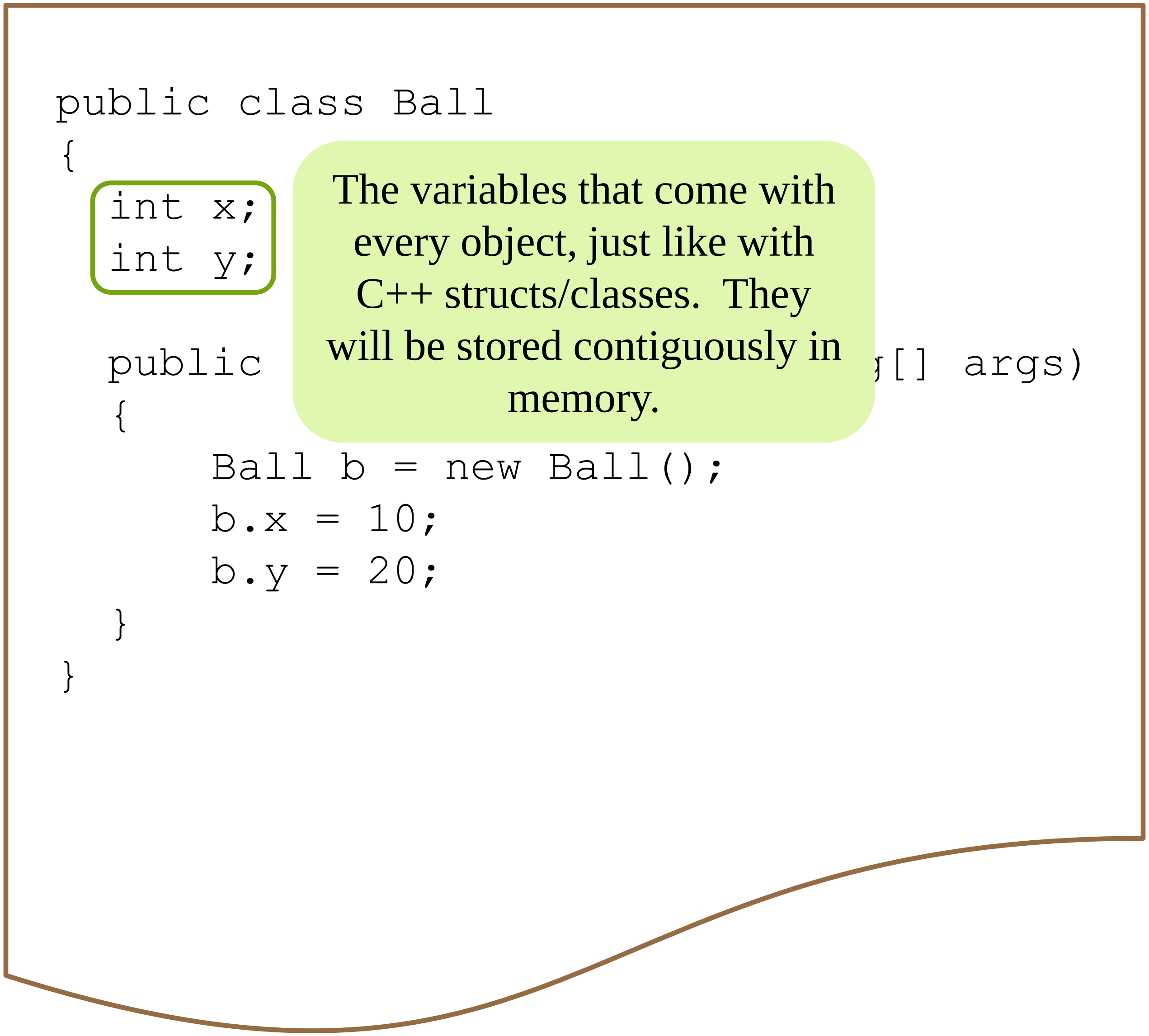
Note that the Java convention is to capitalize the class names, unlike in C++.

Ball.java

```
public class Ball
{

   public static void main(String[] args)
   {
        Ball b = new Ball();
        b.x = 10;
        b.y = 20;
   }
}
```

This keyword has a similar meaning as it does in C++ classes; we will learn it later.

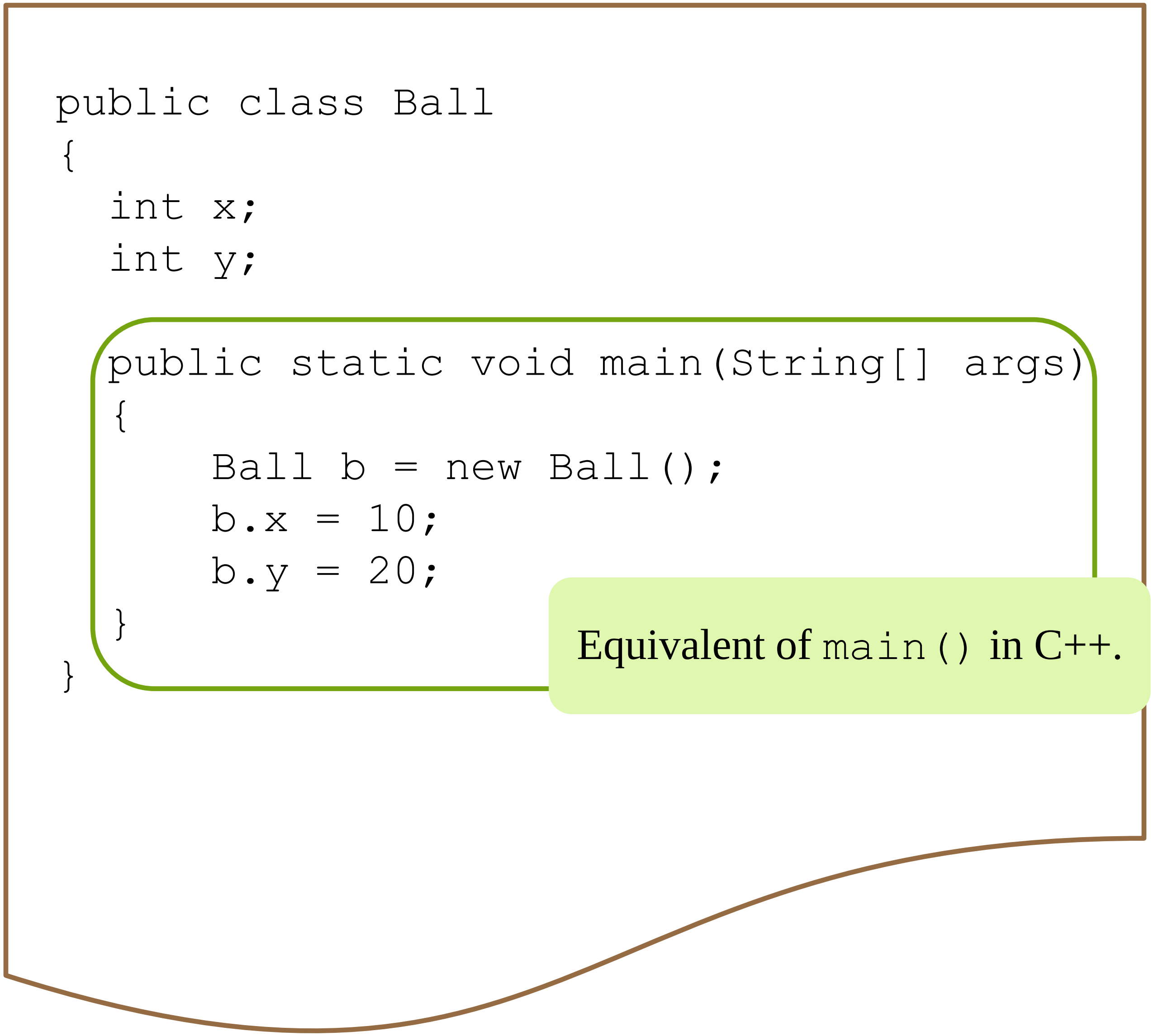
Ball.java



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

  public                              g[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

The variables that come with every object, just like with C++ structs/classes. They will be stored contiguously in memory.

Ball.java



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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

Equivalent of `main()` in C++.

Ball.java

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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

Declaring a variable by saying `Ball b;` does not create space in memory for the object like it does in C++ – you have to use `new Ball()` for that.

Ball.java

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

  public static void main(String[] args)
  {
      Ball b = new Ball();
      b.x = 10;
      b.y = 20;
  }
}
```

After creating a new Ball in memory, you can assign values to its variables just like a struct/class in C++.