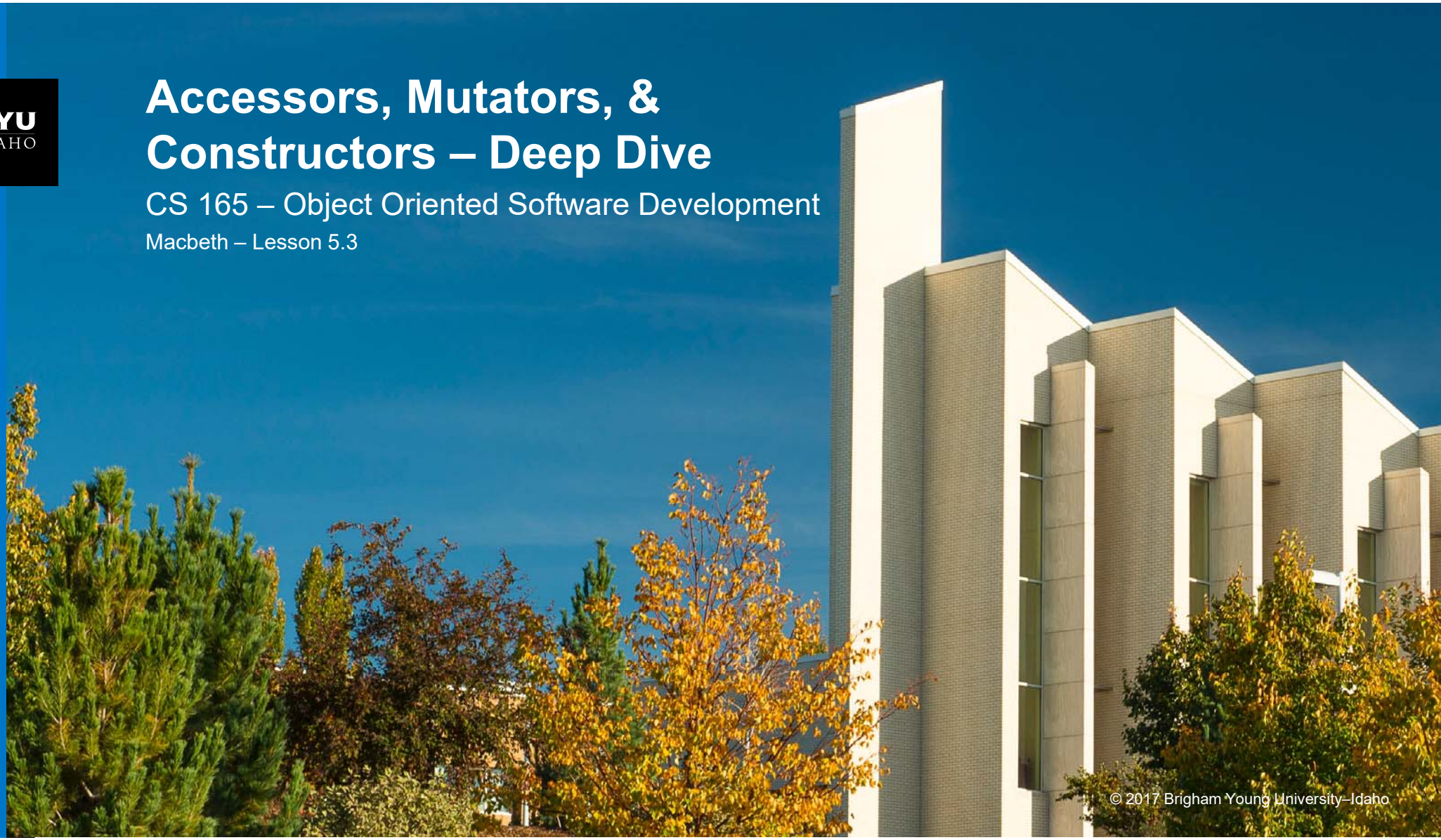
BYU IDAHO

Accessors, Mutators, & Constructors – Deep Dive

CS 165 – Object Oriented Software Development

Macbeth – Lesson 5.3



Agenda

- Opening Prayer
- Music Friday
- Deep Dive of Accessors, Mutators, & Constructors
- Q&A
  - Review Checkpoint B
  - Assignment 05
- Looking Forward

BYU IDAHO

Music Friday

Let Us All Press On (Hymn 243)

Let us all press on in the work of the Lord,
That when life is o'er we may gain a reward;
In the fight for right let us wield a sword,
The mighty sword of truth.

Fear not, though the enemy deride;
Courage, for the Lord is on our side.
We will heed not what the wicked may say,
But the Lord alone we will obey.

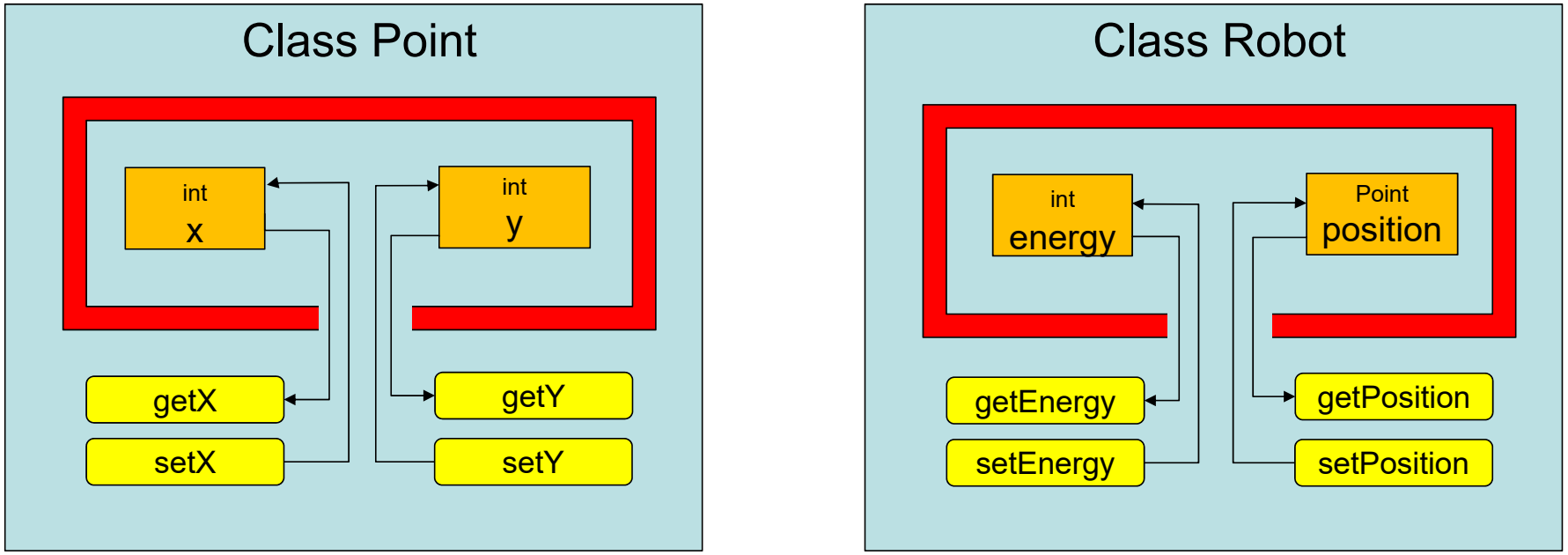
Team Activity

Topics that Need More Attention (from your quiz input):

- Dataflow in a class (member data, getters, setters, public, private)
- Purpose of get functions (accessors) and set functions (mutators)
- Difference between default and non-default constructor
- Initializers in Constructors
- `this`
- Inline Functions

BYU IDAHO

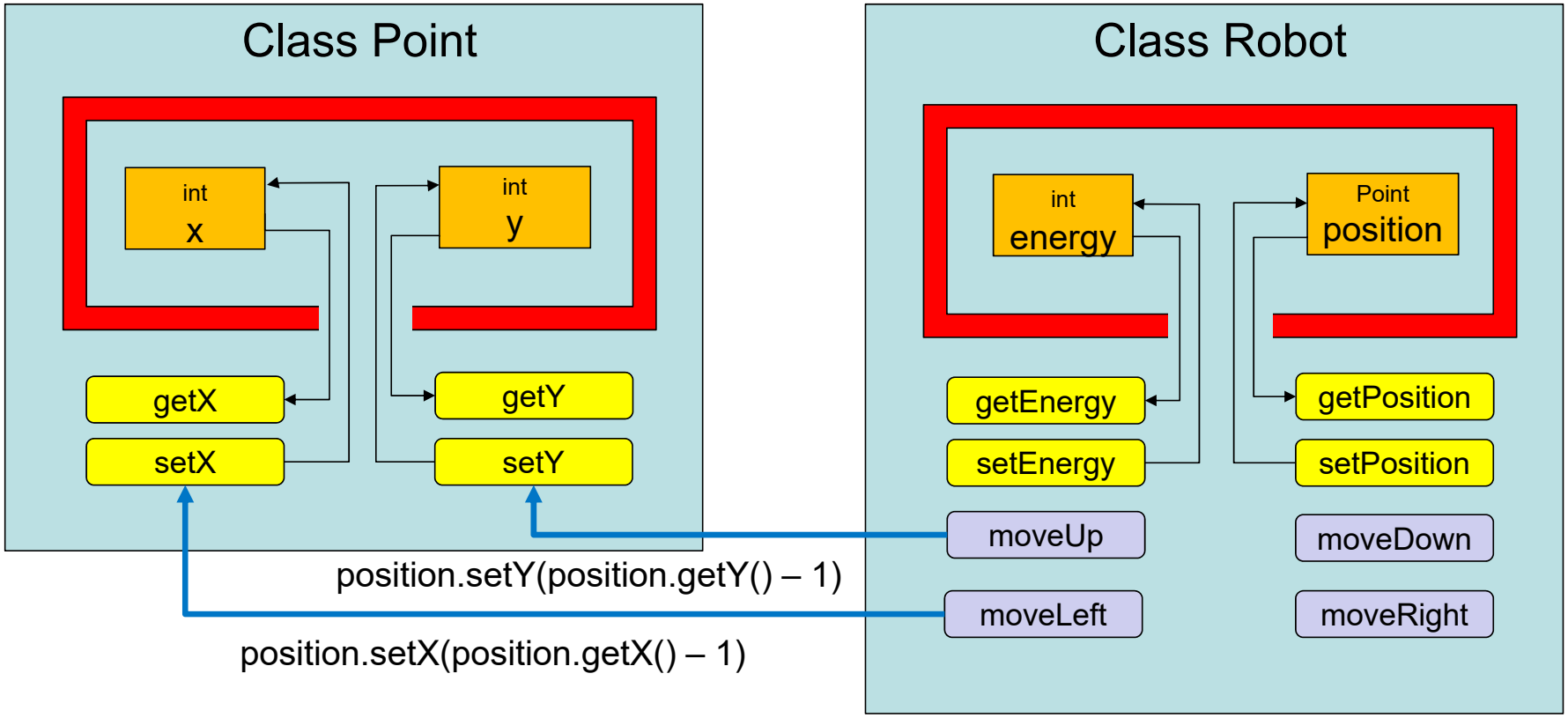
Data Flow in a Class



- Setters provide safe write access to data for the class, other classes, and other non-class functions (e.g. main).
- Getters provide read-only access to data for other classes and non-class functions (e.g. main).

BYU IDAHO

Data Flow in a Class



Constructors

Type	Description	Example

Default	The class will initialize the member data with default values when the object is created.
	
	These constructors do not have parameters.
	
	It is a good practice to always define a default constructor for your class.
		`Point::Point()`
		`{`
		`  setX(5);  // 5 is my default`
		`  setY(5);`
		`}`
		
		`int main()`
		`{`
		`  Point p;`
		`}`

Non-Default	The class will initialize the member data with the values provided from the calling function when the object is created.
	
	These constructors do have parameters.
	
	You can have one or more, but they are optional.
		`Point::Point(int x, int y)`
		`{`
		`  setX(x);  // Use value from user`
		`  setY(y);`
		`}`
		
		`int main()`
		`{`
		`  Point p(3,2);`
		`}`

Initializers

- When you implement a constructor, there are two ways to set the member variables:
  - Assignment statements in the constructor (using equal signs)
  - Initializer List
- The following member variables must be set in the constructor using the initializer list:
  - Constant Member Data (anything with a `const` in front)
  - An object of another class that needs to call a non-default constructor (something with parameters)

Header File

```
class Robot
{
   private:
      Point point;
      int energy;
      const int maxEnergy;

   public:
      Robot();
      Robot(int x, int y);
 };
```

Implementation File

```
Robot::Robot() : maxEnergy(500)
{
   // point will use the default constructor in the Point class
   // maxEnergy must be in the initializer because it's a const
   energy = 0;  // This could be in the initializer
}

Robot::Robot(int x, int y) : maxEnergy(500), point(x,y)
{
   // point with use the non-default constructor in the Point class
   // maxEnergy must be in the initializer because it's a const
   energy = 0; // This could be in the initializer
```

BYU IDAHO

this

- When you see "`this->`" then you know that what follows it is the member data in the object.
- This is used when there are two variables in a function with the same name and you need to tell the compiler which one is which.

```
class Robot
{
   private:
      int energy
   …
}

void setEnergy(int energy)
{
   this->energy    =    energy;
}
```

member data called energy

local variable or parameter in this function

Inline

- There are 2 reasons for creating an inline function:
  - Performance - You want to avoid a function call for constructors, accessors, and mutators that only do a single return or very few assignment statements.
  - Convenience – Since the most common way to make a function inline is to put the implementation in the class declaration (the .h file), you can avoid writing extra code in the class implementation (the .cpp file). The down side is you mix implementation in between the .h and the .cpp file.

Inline

```
class Point
{
   private:
      int x;
   public:
      int getX() const   // This is inline
      {                  // because the
         return x;       // implementation is in
      }                  // the header file.
};
```

Not Inline

```
class Point
{
 private:
      int x;
   public:
      int getX() const;
};
```

```
int Point::getX() const  // In cpp file
{
   return x;
}
```

Looking Forward

- Monday
  - Assignment 5 – E-Commerce Product Inventory (Part 2)
  - Last chance to turn in Checkpoints 5A and 5B
  - Last chance to complete Team Project work and submit quiz

BYU IDAHO