University of Technology, Jamaica
School of Computing and Information Technology

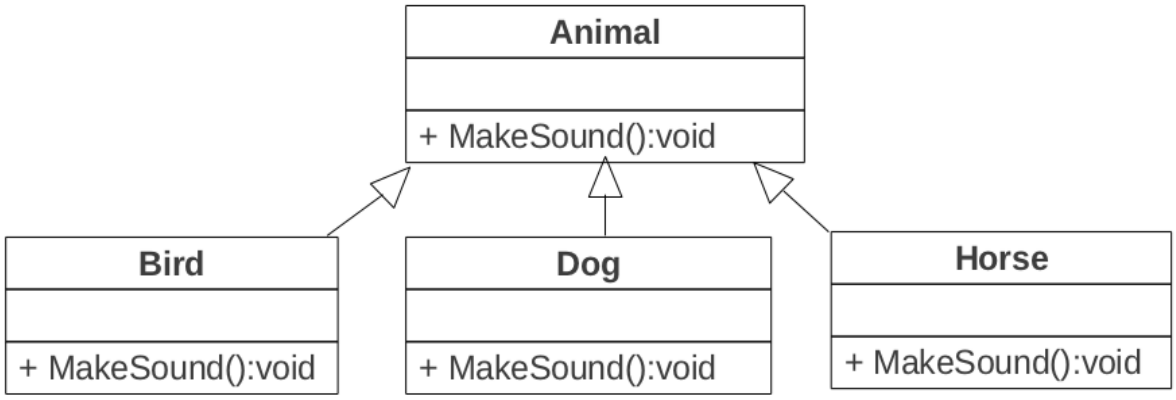
OOP Principles Lab Practical Seven - Polymorphism

Objective:
The objectives of this lab practical are to allow students to be able to:

- practice implementing polymorphism in C++ and Java
- Use a method parameter to achieve polymorphism

Exercise One

Implement the classes depicted in the UML class diagram below, using C++ and/or Java. All the methods and classes are concrete. Write a driver file to demonstrate polymorphism using all the classes in the diagram.



Exercise Two

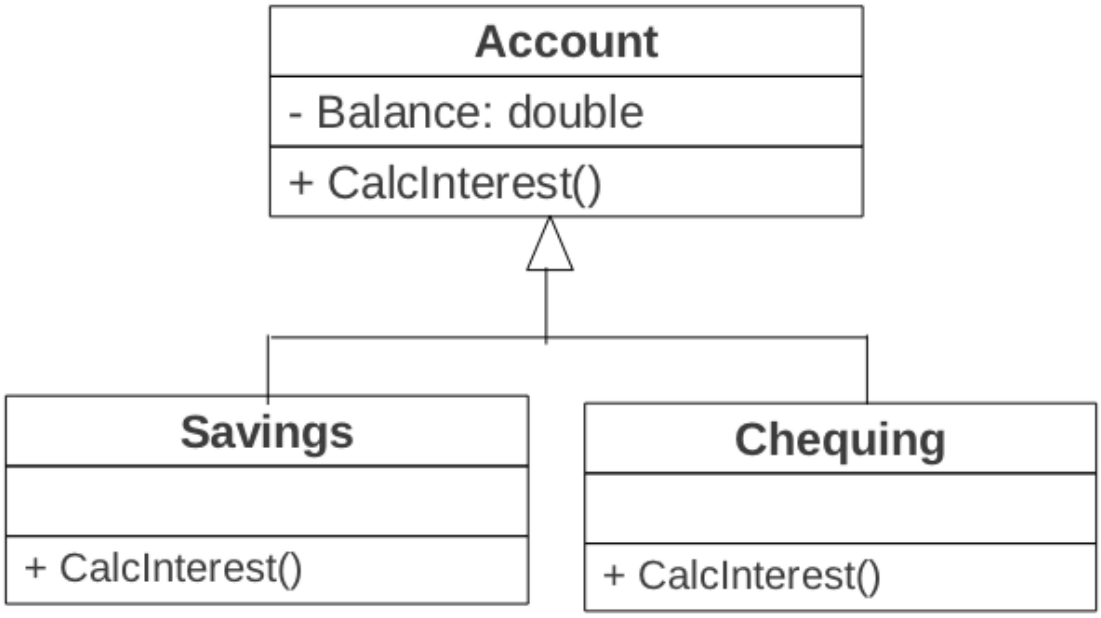
Make the base class abstract. Do not delete any code – only comment out code no longer needed or illegal. What do you need to chance in the driver file to allow it to compile and run now? Again, only comment out code no longer needed or illegal.

Exercise Three

Write a method in your driver file called ProcessSound(). It should accept an object of the base class as a parameter, and return no value. Use it to call a single method to demonstrate polymorphism, based on the argument passed into the method. Show how you would call the method from main to prove that polymorphism is at work.

Exercise Four – homework

Perform exercises one to four above using the UML diagram for Account and other classes shown below.



Answers to exercise one to four

Note: Enter each class and driver in a separate file.

Answer to Exercise One

Implement the classes depicted in the UML class diagram below, using C++ and/or Java. All the methods and classes are concrete. Write a driver file to demonstrate polymorphism using all the classes in the diagram.

C++

```
//Animal class
#ifndef AnimalCl
#define AnimalCl
#include <iostream>
using namespace std;
class Animal
{
	public:
	virtual void MakeSound();
	{
		cout << "MakeSound() in Animal" << endl;
	}
} ;
#endif

#ifndef DogCl
#define DogCl
#include "Animal.h"
#include <iostream>
using namespace std;
class Dog : public Animal
{
	public:
	void MakeSound()
	{
		cout << "MakeSound() in Dog" << endl;
	}
};
#endif
```

```
#ifndef BirdCl
#define BirdCl
#include "Animal.h"
#include <iostream>
using namespace std;
class Bird : public Animal
{
	public:
	void MakeSound()
	{
		cout << "MakeSound() in Bird" << endl;
	}
};
#endif

#ifndef HorseCl
#define HorseCl
#include "Animal.h"
#include <iostream>
using namespace std;
class Horse : public Animal
{
	public:
	void MakeSound()
	{
		cout << "MakeSound() in Horse" << endl;
	}
};
#endif
```

```
#include "Animal.h"
#include "Bird.h"
#include "Dog.h"
#include "Horse.h"

int main()
{

	//object variable can reference parent class Animal
	//and its child classes
	Animal *obj;

	obj = new Animal;
	obj->MakeSound();

	obj = new Bird;
	obj->MakeSound();

	obj = new Dog;
	obj->MakeSound();

	obj = new Horse;
	obj->MakeSound();

	return 0;
}
```

Java

```
//Animal class
public class Animal
{
	public void MakeSound();
	{
		System.out.println("MakeSound() in Animal");
	}
}

public class Bird extends Animal
{
	public void MakeSound()
	{
		System.out.println("MakeSound() in Bird");
	}
}

public class Dog extends Animal
{
	public void MakeSound()
	{
		System.out.println("MakeSound() in Dog");
	}
}
```

```
public class Horse extends Animal
{
	public void MakeSound()
	{
		System.out.println("MakeSound() in Horse");
	}
}

//AnimalDriver to demonstrate polymorphism
public class AnimalDriver
{
	public static void ProcessSound(Animal oRef)
	{
		oRef.MakeSound();
	}
	public static void main(String[] args)
	{
		Animal obj;

		obj = new Bird();
		ProcessSound(obj);

		obj = new Dog();
		ProcessSound(obj);

		obj = new Horse();
		ProcessSound(obj);
	}
}
```

```
	public static void main(String[] args)
	{
		//object variable can reference parent class Animal
		//and its child classes
		Animal obj;

		obj = new Animal();
		obj.MakeSound();

		obj = new Bird();
		obj.MakeSound();

		obj = new Dog();
		obj.MakeSound();

		obj = new Horse();
		obj.MakeSound();

	}
```

Answer to Exercise Two

Make the base class abstract. Do not delete any code – only comment out code no longer needed or illegal. What do you need to chance in the driver file to allow it to compile and run now? Again, only comment out code no longer needed or illegal.

C++

```
//Animal class
#ifndef AnimalCl
#define AnimalCl
#include <iostream>
using namespace std;
class Animal
{
	public:
	virtual void MakeSound() = 0;
//	{
//		cout << "MakeSound() in Animal" << endl;
//	}
} ;
#endif
```

```
int main()
{

	//object variable can reference parent class Animal
	//and its child classes
	Animal *obj;

	//obj = new Animal;
	//obj->MakeSound();

	obj = new Bird;
	obj->MakeSound();

	obj = new Dog;
	obj->MakeSound();

	obj = new Horse;
	obj->MakeSound();

	return 0;
}
```

Java

```java
//Animal class
public abstract class Animal
{
	public abstract void MakeSound();
//	{
//		System.out.println("MakeSound() in Animal");
//	}
}

	public static void main(String[] args)
	{
		//object variable can reference parent class Animal
		//and its child classes
		Animal obj;

		//obj = new Animal();
		//obj.MakeSound();

		obj = new Bird();
		obj.MakeSound();

		obj = new Dog();
		obj.MakeSound();

		obj = new Horse();
		obj.MakeSound();

	}
```

Answer to Exercise Three

Write a method in your driver file called ProcessSound(). It should accept an object of the base class as a parameter, and return no value. Use it to call a single method to demonstrate polymorphism, based on the argument passed into the method. Show how you would call the method from main to prove that polymorphism is at work.

C++

```
#include "Animal.h"
#include "Bird.h"
#include "Dog.h"
#include "Horse.h"

void ProcessSound(Animal *oPtr)
{
	oPtr->MakeSound();
}

int main()
{
	Animal *obj;

	obj = new Bird;
	ProcessSound(obj);

	obj = new Dog;
	ProcessSound(obj);

	obj = new Horse;
	ProcessSound(obj);

	return 0;
}
```

Java

```java
//AnimalDriver to demonstrate polymorphism
public class AnimalDriver
{
	public static void ProcessSound(Animal oRef)
	{
		oRef.MakeSound();
	}
	public static void main(String[] args)
	{
		Animal obj;

		obj = new Bird();
		ProcessSound(obj);

		obj = new Dog();
		ProcessSound(obj);

		obj = new Horse();
		ProcessSound(obj);
	}
}
```