1. There seems to be a semicolon missing from the class projectile’s last bracket. To fix

#ifndef PROJECTILE\_H  
#define PROJECTILE\_H  
class Projectile {  
private:  
double position;  
double speed;  
public:  
Projectile(double speed, double velocity);  
virtual ̃Projectile();  
double getSpeed() const;  
double getVelocity() const;  
}; // end of Projecile class  
#endif

2. This.x and this.y is not the correct syntax for c++. (\*this).x is equivalent to this->x whereas “this” is a pointer to an object.

Second parameter should be position, to keep consistent with the header file, and the way it is coded (this.x, this.y in java syntax).

Changes made were: second argument in parameter, this.x, and this.y equals speed, and position.

Projectile(int speed, int position) {  
this->speed = speed;

this->position = position;  
} // end of constructor

3.

(a) int\* method(int\* arg);

This is a method called method, that takes in a pointer to an int as the parameter. When we return, it returns a pointer to an int.

(b) const int\* method(int\* arg);

This is a method called method, that takes in a pointer to an int. When the method returns, it returns a pointer to a constant int. The method can freely modify the object.

(c) const int\* const method(int\* arg)

This is a method that is constant, that takes in a variable that when is dereferenced, returns a pointer to a constant integer. you cannot modify the data in the address that the pointer points to (referring to the returned pointer).

(d) const int\* const method(const int\* arg);

This is a method that is constant, that takes in a variable that when is dereferenced, returns a pointer to a constant integer. you cannot modify the data in the address that the pointer points to (referring to the returned pointer). The method is not able to modify the pointer variable in the parameter.

(e) const int\* const method(const int\* arg) const;

This is a method that is constant, that takes in a variable that when is dereferenced, returns a pointer to a constant integer. you cannot modify the data in the address that the pointer points to (referring to the returned pointer). The method is not able to modify the pointer variable in the parameter. This method will not modify the object the method is within.

4. C Strings are better than c++ Strings because the performance (speed wise) is better than c++. C strings can be used in c++ code, and in c code; possibly some apis or libraries might use them over c++ strings.

C++ Strings are better than c strings because it has methods, c++ String is a class. You are able to concatenate with c++ strings, with c strings you cannot. C++ Strings also have a length method, which is a lot easier than to calculate the length of a c string. C++ Strings are an object; as opposed to the c strings which are arrays. Thus, comparing strings are easier to write in c++, using c++ strings. C has a fixed limit on string size, whereas c++, you can keep concatenating without worrying about a size limit.

5. A pointer can be reassigned/changed, while a reference variable cannot be reassigned/changed. Pointers can be null, while a reference has to be a non null value. You need to manually dereference pointers to get the value, whereas references automatically dereferenced the variable for you automatically. References need to be assigned a value when initialized, whereas pointers do not.

6.

The deconstructor is called, when the class gets destroyed. Here we are able to close any heap data we might have initiated in the class the deconstructor was called in.