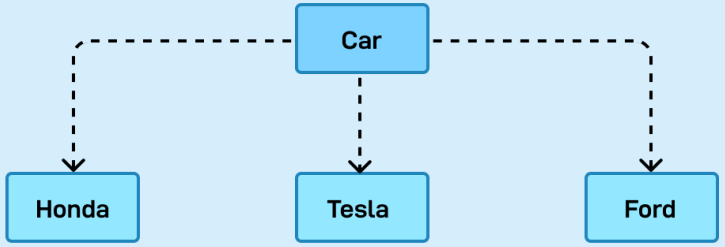
Review on OOP

Object-Oriented Programming

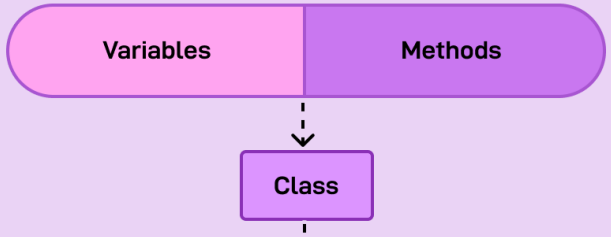
Abstraction

- Process of hiding implementation details and exposing only the functionality to the user.



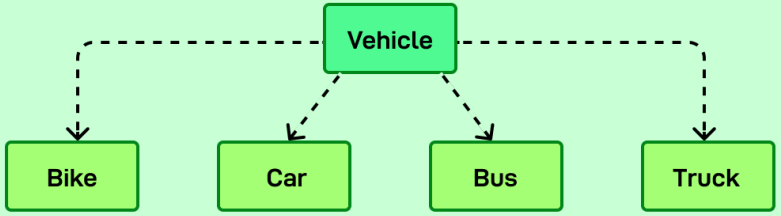
Encapsulation

- Process of wrapping code and data together into a single unit. It means each object in the code should control its own state



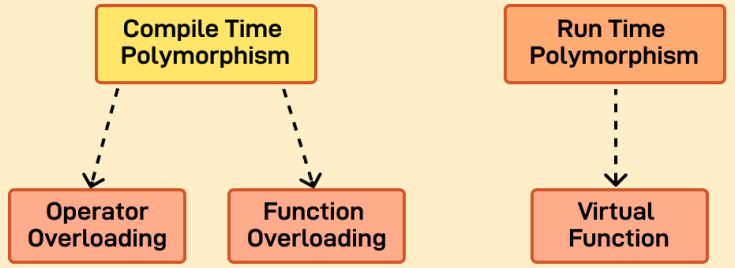
Inheritance

- Process of one class inheriting properties and methods from another class. Used to represent the IS-A relationship between objects



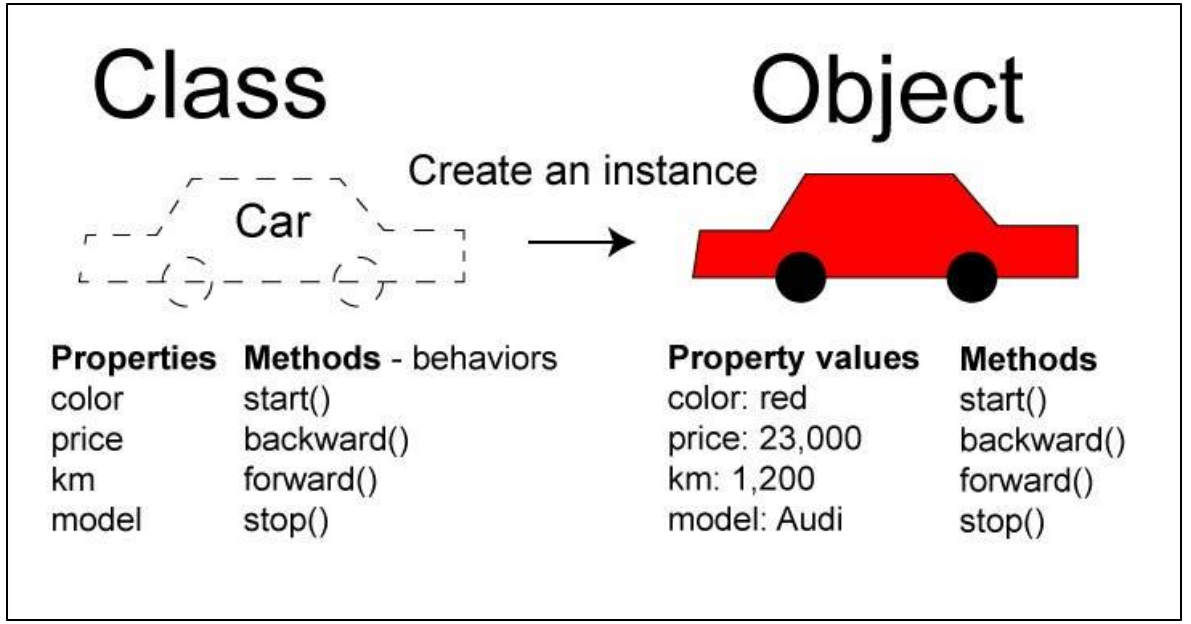
Polymorphism

- Ability to perform many things in many ways. When two types share an inheritance chain, they can be used interchangeably with no errors. This also means using parents like their children



Object-Oriented Programming

- Object-oriented (OO) is a programming framework that structures software around objects.
- These objects can be thought of as real-world items, with their own properties (variables) and behaviors (functions or methods).
- OO programming uses classes as blueprints to create objects.



Generic Examples

- Object-oriented (OO) is a programming framework that structures software around objects.
- These objects can be thought of as real-world items, with their own properties (variables) and behaviors (functions or methods).
- OO programming uses <u>classes</u> as blueprints to create objects.

Class Student	Class Teacher	Class Exam
Properties	**Properties**	**Properties**
First Name	First Name	Exam Date
Last Name	Last Name	Exam Subject
Social Sec No	Social Sec No	Subject Code
Class Standard	Subject	Result Date
Email	Department	No Of Students
Methods	**Methods**	**Methods**
readBook ( );	setPaper ( );	checkPaper ( );
giveExam ( );	giveLecture ( );	publishResult ( );
calculatePresenty ( );	calculateScore ( );	calculateResult ( );

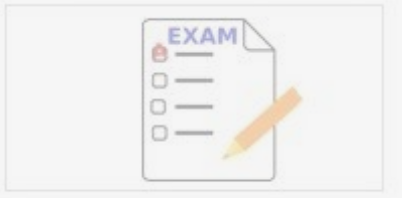
What are other examples?

Generic Examples

```cpp
#include <iostream>
using namespace std;

class Student {
public:
    string FirstName;
    string LastName;
    string SocialSecNo;
    string ClassStandard;
    string Email;

    void readBook(){
        cout << FirstName << " " << LastName << " is reading a book." << endl;
    }

    void giveExam(){
        cout << FirstName << " " << LastName << " is giving an exam." << endl;
    }

    void calculatePresenty(){
        cout << FirstName << " " << LastName << " is calculating presenty." << endl;
    }
};

int main() {
    Student student1;
    student1.FirstName = "John";
    student1.LastName = "Doe";
    student1.SocialSecNo = "123-45-6789";
    student1.ClassStandard = "10th Grade";
    student1.Email = "john.doe@example.com";

    student1.readBook();
    student1.giveExam();
    student1.calculatePresenty();

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Run

```
John Doe is reading a book.
John Doe is giving an exam.
John Doe is calculating presenty.
 *  Terminal will be reused by tasks, press any key to close it.
```



Class Student	Class Teacher	Class Exam
Properties	**Properties**	**Properties**
First Name	First Name	Exam Date
Last Name	Last Name	Exam Subject
Social Sec No	Social Sec No	Subject Code
Class Standard	Subject	Result Date
Email	Department	No Of Students
Methods	**Methods**	**Methods**
readBook ( ) ;	setPaper ( ) ;	checkPaper ( ) ;
giveExam ( ) ;	giveLecture ( ) ;	publishResult ( ) ;
calculatePresenty ( ) ;	calculateScore ( ) ;	calculateResult ( ) ;

Generic Examples

```cpp
#include <iostream>
using namespace std;

class Teacher {
public:
    string FirstName;
    string LastName;
    string SocialSecNo;
    string Subject;
    string Email;

    void setPaper(){
        cout << FirstName << " " << LastName << " is setting a paper." << endl;
    }

    void giveLecture(){
        cout << FirstName << " " << LastName << " is giving a lecture." << endl;
    }

    void calculateScore(){
        cout << FirstName << " " << LastName << " is calculating score." << endl;
    }
};

int main() {
    Teacher teacher1;
    teacher1.FirstName = "John";
    teacher1.LastName = "Doe";
    teacher1.SocialSecNo = "123-45-6789";
    teacher1.Subject = "Mathematics";
    teacher1.Email = "john.doe@example.com";

    teacher1.setPaper();
    teacher1.giveLecture();
    teacher1.calculateScore();

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

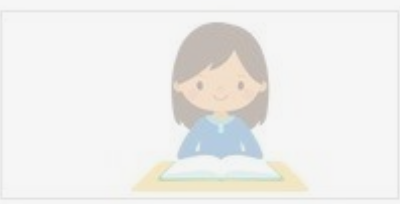
```
John Doe is setting a paper.
John Doe is giving a lecture.
John Doe is calculating score.
* Terminal will be reused by tasks, press any key to close it.
```

Generic Examples

```cpp
#include <iostream>
using namespace std;

class Exam {
public:
    string ExamDate;
    string ExamSubject;
    string SubjectCode;
    string ResultDate;
    string NoOfStudents;

    void checkPaper(){
        cout << "Checking paper for " << NoOfStudents << " students." << endl;
    }

    void publishResults(){
        cout << "Publishing results for " << NoOfStudents << " students." << endl;
    }

    void calculateresult(){
        cout << "Calculating results for " << NoOfStudents << " students." << endl;
    }
};

int main() {
    Exam exam1;
    exam1.ExamDate = "2023-05-01";
    exam1.ExamSubject = "Mathematics";
    exam1.SubjectCode = "6789";
    exam1.ResultDate = "2023-05-15";
    exam1.NoOfStudents = "30";

    exam1.checkPaper();
    exam1.publishResults();
    exam1.calculateresult();

    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Run

```
Checking paper for 30 students.
Publishing results for 30 students.
Calculating results for 30 students.
* Terminal will be reused by tasks, press any key to close it.
```



Class Student	Class Teacher	Class Exam
Properties	Properties	Properties
First Name	First Name	Exam Date
Last Name	Last Name	Exam Subject
Social Sec No	Social Sec No	Subject Code
Class Standard	Subject	Result Date
Email	Department	No Of Students
Methods	Methods	Methods
readBook ( ) ;	setPaper ( ) ;	checkPaper ( ) ;
giveExam ( ) ;	giveLecture ( ) ;	publishResult ( ) ;
calculatePresenty ( ) ;	calculateScore ( ) ;	calculateResult ( ) ;

Example 3 from Week 8

Example 3: Defining Classes And Creating Objects

As the example, consider TV sets. Each TV is an object with state (current channel, current volume level, power on or off) and behaviors (change channels, adjust volume, turn on/off). You can use a class to model TV sets. The UML diagram for the class is shown below.

TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
+TV()	Constructs a default TV object.
+turnOn(): void	Turns on this TV.
+turnOff(): void	Turns off this TV.
+setChannel(newChannel: int): void	Sets a new channel for this TV.
+setVolume(newVolumeLeve: int): void	Sets a new volume level for this TV.
+channelUp(): void	Increases the channel number by 1.
+channelDown(): void	Decreases the channel number by 1.
+volumeUp(): void	Increases the volume level by 1.
+volumeDown(): void	Decreases the volume level by 1.

Example 3 from Week 8

```cpp
#include <iostream>
using namespace std;

class TV{
public:
  int channel;
  int volumeLevel;
  bool on;

  TV(){
    channel = 1;
    volumeLevel = 1;
    on = false;
  }

  void turnOn(){ on = true; }
  void turnOff(){ on = false; }
  void setChannel(int newChannel){
    if (on && newChannel >= 1 && newChannel <= 120)
      channel = newChannel;
  }
  void setVolume(int newVolumeLevel){
    if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
      volumeLevel = newVolumeLevel;
  }
  void channelUp(){ if (on && channel < 120) channel++; }
  void channelDown(){ if (on && channel > 1) channel--; }
  void volumeUp(){ if (on && volumeLevel < 7) volumeLevel++; }
  void volumeDown(){ if (on && volumeLevel > 1) volumeLevel--; }
};

int main(){
  TV tv1;
  tv1.turnOn();
  tv1.setChannel(30);
  tv1.setVolume(3);

  cout << "tv1's channel is " << tv1.channel
    << " and volume level is " << tv1.volumeLevel << endl;
  return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

* Executing task: /bin/zsh -c ./build/Debug/outDebug

tv1's channel is 30 and volume level is 3
* Terminal will be reused by tasks, press any key to close it.
```

TV
channel: int volumeLevel: int on: bool
+TV() +turnOn(): void +turnOff(): void +setChannel(newChannel: int): void +setVolume(newVolumeLeve: int): void +channelUp(): void +channelDown(): void +volumeUp(): void +volumeDown(): void

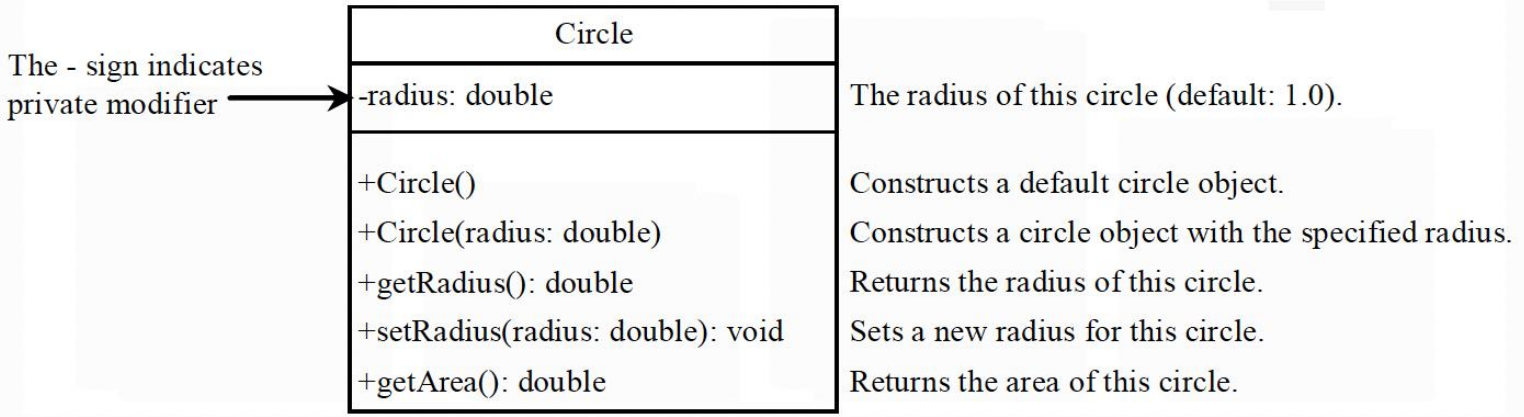
The current channel (1 to 120) of this TV.
The current volume level (1 to 7) of this TV.
Indicates whether this TV is on/off.

Constructs a default TV object.
Turns on this TV.
Turns off this TV.
Sets a new channel for this TV.
Sets a new volume level for this TV.
Increases the channel number by 1.
Decreases the channel number by 1.
Increases the volume level by 1.
Decreases the volume level by 1.

Example 4 from Week 8

Example 4: New Circle Class

Let us create a new circle class with a private data field radius and its associated accessor and mutator functions. The class diagram is shown in Figure below.



Example 4 from Week 8

TestCircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > TestCircleWithPrivateDataFields.cpp > ...

```cpp
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;
int main()
{
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() << endl;

        cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    return 0;
}
```

CircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > CircleWithPrivateDataFiel

```cpp
#include "CircleWithPrivateDataFields.h"
// Construct a default circle object
Circle::Circle(){
    radius = 1;
}
// Construct a circle object
Circle::Circle(double newRadius){
    radius = newRadius;
}
// Return the area of this circle
double Circle::getArea(){
    return radius * radius * 3.14159;
}
// Return the radius of this circle
double Circle::getRadius(){
    return radius;
}
// Set a new radius
void Circle::setRadius(double newRadius){
    radius = (newRadius >= 0) ? newRadius : 0;
}
```

CircleWithPrivateDataFields.h

Week 8 > Lab > Codes > Lab8-4 > Cir

```cpp
#ifndef CIRCLE_H
#define CIRCLE_H
class Circle
{
public:
    Circle();
    Circle(double);
    double getArea();
    double getRadius();
    void setRadius(double);
private:
    double radius;
};
#endif
```

```cpp
#include <iostream>
using namespace std;

class Circle{
public:
    // Construct a default circle object
    Circle(){
        radius = 1;
    }
    // Construct a circle object
    Circle(double newRadius){
        radius = newRadius;
    }
    // Return the area of this circle
    double getArea(){
        return radius * radius * 3.14159;
    }
    // Return the radius of this circle
    double getRadius(){
        return radius;
    }
    // Set a new radius
    void setRadius(double newRadius){
        radius = (newRadius >= 0) ? newRadius : 0;
    }

private:
    double radius;

};

int main(){
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() << endl;

    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
The area of the circle of radius 1 is 3.14159
The area of the circle of radius 5 is 78.5397
The area of the circle of radius 100 is 31415.9
* Terminal will be reused by tasks, press any key to close it.
```

Example 4 from Week 8

TestCircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > TestCircleWithPrivateDataFields.cpp > ...

```cpp
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;
int main()
{
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() << endl;

        cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    return 0;
}
```

CircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > CircleWithPrivateDataFiel

```cpp
#include "CircleWithPrivateDataFields.h"
// Construct a default circle object
Circle::Circle(){
    radius = 1;
}
// Construct a circle object
Circle::Circle(double newRadius){
    radius = newRadius;
}
// Return the area of this circle
double Circle::getArea(){
    return radius * radius * 3.14159;
}
// Return the radius of this circle
double Circle::getRadius(){
    return radius;
}
// Set a new radius
void Circle::setRadius(double newRadius){
    radius = (newRadius >= 0) ? newRadius : 0;
}
```

CircleWithPrivateDataFields.h

Week 8 > Lab > Codes > Lab8-4 > Cir

```cpp
#ifndef CIRCLE_H
#define CIRCLE_H
class Circle
{
public:
    Circle();
    Circle(double);
    double getArea();
    double getRadius();
    void setRadius(double);
private:
    double radius;
};
#endif
```

When we use one file, we don't need function prototypes.

```cpp
#include <iostream>
using namespace std;

class Circle{
public:
    // Construct a default circle object
    Circle(){
        radius = 1;
    }
    // Construct a circle object
    Circle(double newRadius){
        radius = newRadius;
    }
    // Return the area of this circle
    double getArea(){
        return radius * radius * 3.14159;
    }
    // Return the radius of this circle
    double getRadius(){
        return radius;
    }
    // Set a new radius
    void setRadius(double newRadius){
        radius = (newRadius >= 0) ? newRadius : 0;
    }

private:
    double radius;

};

int main(){
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() << endl;

    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    return 0;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
The area of the circle of radius 1 is 3.14159
The area of the circle of radius 5 is 78.5397
The area of the circle of radius 100 is 31415.9
* Terminal will be reused by tasks, press any key to close it.
```

Compile and run code with multiple files:

```
(base) netzahdzc@Lab8-4 % g++ TestCircleWithPrivateDataFields.cpp CircleWithPrivateDataFields.cpp -o test
(base) netzahdzc@Lab8-4 % ./test
```

Example 4 from Week 8

TestCircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > TestCircleWithPrivateDataFields.cpp > ...

```cpp
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;
int main()
{
    Circle circle1;
    Circle circle2(5.0);

    cout << "The area of the circle of radius "
        << circle1.getRadius() << " is " << circle1.getArea() << endl;

        cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    // Modify circle radius
    circle2.setRadius(100);
    cout << "The area of the circle of radius "
        << circle2.getRadius() << " is " << circle2.getArea() << endl;
    return 0;
}
```

CircleWithPrivateDataFields.cpp

Week 8 > Lab > Codes > Lab8-4 > CircleWithPrivateDataFiel

```cpp
#include "CircleWithPrivateDataFields.h"
// Construct a default circle object
Circle::Circle(){
    radius = 1;
}
// Construct a circle object
Circle::Circle(double newRadius){
    radius = newRadius;
}
// Return the area of this circle
double Circle::getArea(){
    return radius * radius * 3.14159;
}
// Return the radius of this circle
double Circle::getRadius(){
    return radius;
}
// Set a new radius
void Circle::setRadius(double newRadius){
    radius = (newRadius >= 0) ? newRadius : 0;
}
```

CircleWithPrivateDataFields.h

Week 8 > Lab > Codes > Lab8-4 > Cir

```cpp
#ifndef CIRCLE_H
#define CIRCLE_H
class Circle
{
public:
    Circle();
    Circle(double);
    double getArea();
    double getRadius();
    void setRadius(double);
private:
    double radius;
};
#endif
```

The `#ifndef`, `#define`, and `#endif` are **include guards** to prevent the header from being included multiple times.

`#ifndef CIRCLE_H`

Means **"if not defined"** — it checks whether a macro named `CIRCLE_H` has been defined before.

If it **has not** been defined yet, the compiler will **include** the following lines of code.

If it **has** been defined, it skips everything until the `#endif`.

`#define CIRCLE_H`

Defines the macro `CIRCLE_H`, so that the next time this file is included, the condition `#ifndef CIRCLE_H` will fail (because it *is* now defined).

That way, the header's contents are **included only once**.

`#endif`

Marks the end of the guarded section.

Note how the two .cpp files include the header file "CircleWithPrivateDataFields.h"

Physical and non-physical objects, which can be analogously viewed as objects

Physical objects

Class Student	Class Teacher	Class Exam
Properties	**Properties**	**Properties**
First Name	First Name	Exam Date
Last Name	Last Name	Exam Subject
Social Sec No	Social Sec No	Subject Code
Class Standard	Subject	Result Date
Email	Department	No Of Students
Methods	**Methods**	**Methods**
readBook ( );	setPaper ( );	checkPaper ( );
giveExam ( );	giveLecture ( );	publishResult ( );
calculatePresenty ( );	calculateScore ( );	calculateResult ( );

Non-physical objects

BankAccount

Description:
Represents a customer's account in a banking system, storing data such as account number and balance, and providing operations like deposit and withdrawal.

Reason it is Non-Physical:
A bank account is an abstract concept — it does not physically exist but represents a relationship between the customer and the bank, maintained through data and rules.

Class	**BankAccount**
Attributes	- accountNumber: String - balance: float
Methods	+ deposit(amount: float): void + withdraw(amount: float): void

Event

Description:
Represents something that occurs at a specific time, such as a meeting or system-triggered action.

Reason it is Non-Physical:
An event is a conceptual occurrence. It has a name and time, but it has no tangible or physical presence in the real world.

Class	**Event**
Attributes	- name: String - date: Date
Methods	+ getDetails(): String

Example 1 from Week 9

Example 1: The Loan Class

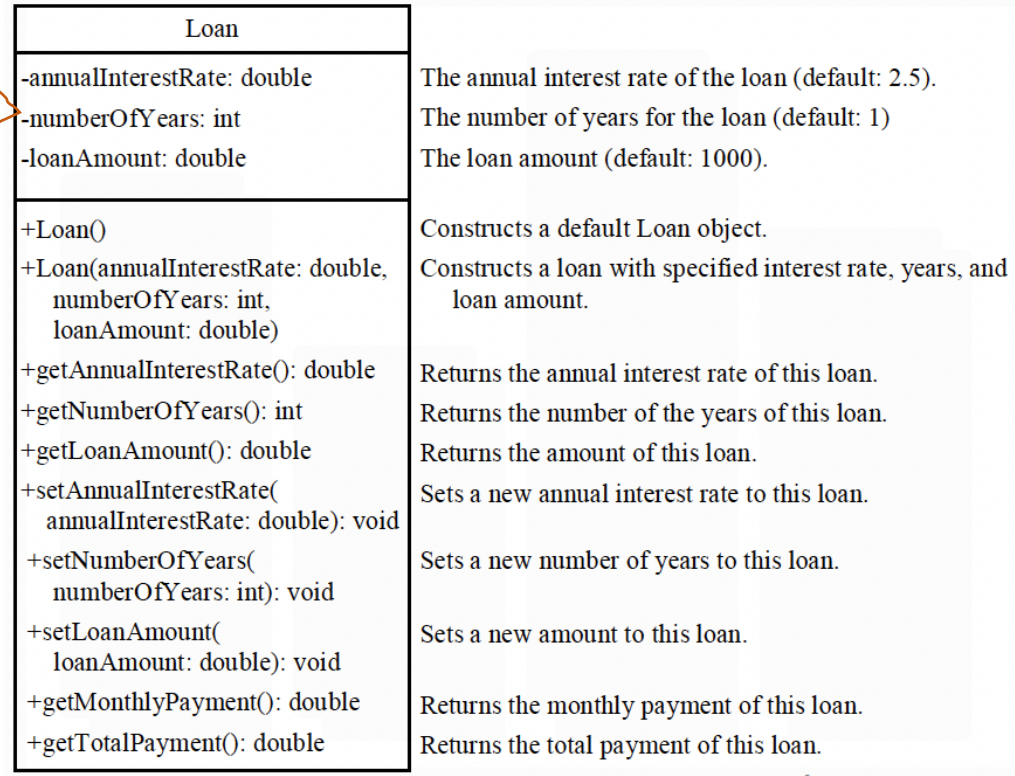
A specific loan can be viewed as an object of a Loan class. Interest rate, loan amount, and loan period are its data properties, and computing monthly payment and total payment are its functions. Let us use the Loan class as an example to demonstrate the creation and use of classes. Loan has the data fields annualInterestRate, numberOfYears, and loanAmount, and the functions getAnnualInterestRate, getNumberOfYears, getLoanAmount, setAnnualInterestRate, setNumberOfYears, setLoanAmount, getMonthlyPayment,and getTotalPayment, as shown in Figure below.

Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000).
+Loan()	Constructs a default Loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+getMonthlyPayment(): double	Returns the monthly payment of this loan.
+getTotalPayment(): double	Returns the total payment of this loan.

Example 1 from Week 9: The Loan Class

Example of a non-physical objects

A specific loan can be viewed as an object of a Loan class. Interest rate, loan amount, and loan period are its data properties, and computing monthly payment and total payment are its functions. Let us use the Loan class as an example to demonstrate the creation and use of classes. Loan has the data fields annualInterestRate, numberOfYears, and loanAmount, and the functions getAnnualInterestRate, getNumberOfYears, getLoanAmount, setAnnualInterestRate, setNumberOfYears, setLoanAmount, getMonthlyPayment,and getTotalPayment, as shown in Figure below.



Example 1 from Week 9: The Loan Class

Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000).
+Loan()	Constructs a default Loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+getMonthlyPayment(): double	Returns the monthly payment of this loan.
+getTotalPayment(): double	Returns the total payment of this loan.

Note how in the main function, we create the object "loan" and then invoke its respective functions to change the state of the object.

Loan.h

```
#ifndef LOAN_H
#define LOAN_H

class Loan{
public:
  Loan();
  Loan(double rate, int years, double amount);
  double getAnnualInterestRate();
  int getNumberOfYears();
  double getLoanAmount();
  void setAnnualInterestRate(double rate);
  void setNumberOfYears(int years);
  void setLoanAmount(double amount);
  double getMonthlyPayment();
  double getTotalPayment();

private:
  double annualInterestRate;
  int numberOfYears;
  double loanAmount;
};

#endif
```

Loan.cpp

```
#include "Loan.h"
#include <cmath>
using namespace std;

Loan::Loan(){
  annualInterestRate = 2.5;
  numberOfYears = 2;
  loanAmount = 1000;
}

Loan::Loan(double rate, int years, double amount){
  annualInterestRate = rate;
  numberOfYears = years;
  loanAmount = amount;
}

double Loan::getAnnualInterestRate(){ return annualInterestRate; }

int Loan::getNumberOfYears(){ return numberOfYears; }

double Loan::getLoanAmount(){ return loanAmount; }

void Loan::setAnnualInterestRate(double rate){ annualInterestRate = rate; }

void Loan::setNumberOfYears(int years){ numberOfYears = years; }

void Loan::setLoanAmount(double amount){ loanAmount = amount; }

double Loan::getMonthlyPayment(){
  double monthlyInterestRate = annualInterestRate / 1200;
  return loanAmount * monthlyInterestRate / (1 -
    (pow(1 / (1 + monthlyInterestRate), numberOfYears * 12)));
}

double Loan::getTotalPayment(){
  return getMonthlyPayment() * numberOfYears * 12;
}
```

TestLoanClass.cpp

```
#include <iostream>
#include <iomanip>
#include "Loan.h"
using namespace std;

int main(){
  // Enter annual interest rate
  cout << "Enter yearly interest rate, for example 8.25: ";
  double annualInterestRate;
  cin >> annualInterestRate;

  // Enter number of years
  cout << "Enter number of years as an integer, for example 5: ";
  int numberOfYears;
  cin >> numberOfYears;

  // Enter loan amount
  cout << "Enter loan amount, for example 120000.95: ";
  double loanAmount;
  cin >> loanAmount;

  // Create Loan object
  Loan loan(annualInterestRate, numberOfYears, loanAmount);

  // Display results
  cout << fixed << setprecision(2);
  cout << "The monthly payment is "
    << loan.getMonthlyPayment() << endl;
  cout << "The total payment is " << loan.getTotalPayment() << endl;

  return 0;
}
```

Compile and run code with multiple files:

```
(base) netzahdzc@Lab9-1 % g++ TestLoanClass.cpp Loan.cpp -o test
(base) netzahdzc@Lab9-1 % ./test
```

Today's lab session:

Make sure to create 3 files per example:

1. One file with the extension **.cpp** to write the main function.
2. Other file with the extension **.cpp** to declare the constructors and functions.
3. One file –the header file– with extension **.h** to write the private properties and function prototypes.

- Run example 1, as provided in this slides.

- Solve example 3, using the provided material.
- Solve example 2, using the next slide's equation.
- Solve example 4.

- Revise example 5, making sure it is fully understood.

Feel free to download this slide using the QR code:

Use the below calculation of BMI and BMI-classification for Example 2:

```
double BMI::getBMI() const
{
  const double KILOGRAMS_PER_POUND = 0.45359237;
  const double METERS_PER_INCH = 0.0254;
  double bmi = weight * KILOGRAMS_PER_POUND /
    ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
  return bmi;
}

string BMI::getStatus() const
{
  double bmi = getBMI();
  if (bmi < 18.5)
    return "Underweight";
  else if (bmi < 25)
    return "Normal";
  else if (bmi < 30)
    return "Overweight";
  else
    return "Obese";
}
```