# LAB CYCLE 3

## CLASSES AND OBJECTS

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**21) Create a Python class called `Library` to manage a library's book collection with the following attributes and methods:**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

- Attributes: `books` (a list of book titles).
- Methods:
  - `__init__(self)` - Initialize an empty list to store books.
  - `add_book(self, book_title)` - Use a list to add a book to the collection.
  - `remove_book(self, book_title)` - Use list methods to remove a book from the collection.
  - `list_books(self)` - Return the list of all books in the collection.

Create a Python program that does the following:

1. Import the `Library` class from a separate module.
2. Create an instance of the `Library` class.
3. Add several books to the library's collection using the `add_book` method.
4. Remove a book from the collection using the `remove_book` method.
5. List all the books in the library's collection using the `list_books` method.
6. Display the results.

**Hint : Use  a list to store the book titles**

**Sample Output**

***Books in the Library:***
***['The Great Gatsby', 'To Kill a Mockingbird', '1984']***

***Books after removal:***
***['The Great Gatsby', '1984']***

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**22) Create a Python program that represents a contact application using a dictionary to store contact information. The application should take user input and have the following functionality:**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1. Initialize an empty dictionary to store contact information.

2. Create functions or methods for the following operations:
   - `add_contact(name, phone_number, email)` - Add a new contact with the given name, phone number, and email to the dictionary.
   - `update_contact(name, new_phone_number, new_email)` - Update the phone number and email of an existing contact with the given name.
   - `delete_contact(name)` - Delete a contact with the given name.

- `list_contacts()` - Display a list of all contacts in the dictionary.

**Sample Output:**

***Contact Application Menu:***
***1. Add Contact***
***2. Update Contact***
***3. Delete Contact***
***4. List Contacts***
***5. Quit***
***Enter your choice: 1***

***Enter the name: Alice***
***Enter the phone number: 123-456-7890***
***Enter the email: alice@example.com***

***Contact Application Menu:***
***1. Add Contact***
***2. Update Contact***
***3. Delete Contact***
***4. List Contacts***
***5. Quit***
***Enter your choice: 4***

***Contacts:***
***Name: Alice, Phone Number: 123-456-7890, Email: alice@example.com***
***Name: Bob, Phone Number: 987-654-3210, Email: bob@example.com***

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**23) Create a Python program to manage a shopping cart items using the following classes and methods:**

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

1. Create an `Item` class with attributes and methods:
   - Attributes: `name`, `price`, `quantity`, `description`, and `manufacturer`.
   - Methods:
     - `__init__(self, name, price, description, manufacturer)` - Initialize the item with a name, price, description, and manufacturer. Quantity should be set to 1 by default.
     - `increase_quantity(self)` - Increase the quantity of the item by 1.
     - `decrease_quantity(self)` - Decrease the quantity of the item by 1.
     - `get_item_details(self)` - Return a string with details of the item.

2. Create a `ShoppingCart` class with the following methods:
   - `__init__(self)` - Initialize an empty list of items.
   - `add_item(self, item)` - Accept an Item object and add it to the cart.
   - `remove_item(self, item)` - Accept an Item object and remove it from the cart.
   - `display_cart(self)` - Display the items in the cart with their details.

Create a menu-driven program for the user to perform the following actions:

- Add an item to the shopping cart.
- Remove an item from the shopping cart.
- Display the contents of the shopping cart.
- Quit the program.

**Sample Output**

***Shopping Cart Menu:***
***1. Add an item to the cart***
***2. Remove an item from the cart***
***3. Display the contents of the cart***
***4. Quit***
***Enter your choice: 3***

***Shopping Cart:***
***Name: Laptop, Price: $1200.0, Quantity: 1, Description: High-performance laptop, Manufacturer: HP***
***Name: Smartphone, Price: $800.0, Quantity: 1, Description: Latest smartphone model, Manufacturer: Apple***

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
**24) Exercise: Vehicle Inheritance**
You are given a base class `Vehicle`, and you need to create three subclasses: `Car`, `Bicycle`, and `Motorcycle`. Each of these subclasses should inherit from the `Vehicle` class and have their own unique properties and methods.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Here are the details for each class:

1. `Vehicle` (Base Class):
   - Properties:
     - `make` (string) - The make or manufacturer of the vehicle.
     - `model` (string) - The model of the vehicle.
     - `year` (int) - The manufacturing year of the vehicle.
     - `fuel` (float) - The current fuel level in liters.
   - Methods:
     - `__init__(self, make, model, year, fuel)` - Initialize the vehicle with the provided make, model, year, and initial fuel level.
     - `refuel(self, liters)` - Refuel the vehicle with the specified number of liters.
     - `drive(self, distance)` - Simulate driving the vehicle for the given distance in kilometers. This should decrease the fuel level and return the remaining fuel.

2. `Car` (Subclass of `Vehicle`):
   - Properties (in addition to the ones inherited from `Vehicle`):

    - `num_doors` (int) - The number of doors on the car.
  - Methods:
    - `__init__(self, make, model, year, fuel, num_doors)` - Initialize the car with the provided make, model, year, fuel level, and number of doors.
    - `honk(self)` - Print a honking message specific to a car.

3. `Bicycle` (Subclass of `Vehicle`):
  - Properties (in addition to the ones inherited from `Vehicle`):
    - `num_gears` (int) - The number of gears on the bicycle.
  - Methods:
    - `__init__(self, make, model, year, fuel, num_gears)` - Initialize the bicycle with the provided make, model, year, fuel level, and number of gears.
    - `ring_bell(self)` - Print a message indicating the ringing of a bell on a bicycle.

4. `Motorcycle` (Subclass of `Vehicle`):
  - Properties (in addition to the ones inherited from `Vehicle`):
    - `engine_displacement` (float) - The engine displacement in cc (cubic centimeters).
  - Methods:
    - `__init__(self, make, model, year, fuel, engine_displacement)` - Initialize the motorcycle with the provided make, model, year, fuel level, and engine displacement.
    - `start_engine(self)` - Print a message indicating the starting of the motorcycle's engine.

Your task is to implement the classes and their methods according to the provided specifications. Then, create instances of each of the three subclasses and demonstrate their functionality by calling their methods.

Your task is to implement the classes and their methods according to the provided specifications. Then, create instances of each of the three subclasses and demonstrate their functionality by calling their methods.

**Sample Output**

Car:
Driving 50 kilometers. Fuel remaining: 25.0 liters
Honk! Honk!

Bicycle:
Driving 10 kilometers. Fuel remaining: 0.0 liters
Ring, ring! (Sound of a bicycle bell)

Motorcycle:
Driving 100 kilometers. Fuel remaining: 0.0 liters
Starting the motorcycle's engine...

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

## 25) Exercise: PhoneBook

Create a base class **Contact** that represents an individual contact, and two subclasses **Person** and **Business** for different types of contacts. Each contact type will have additional properties and methods.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create a base class Contact with the following properties:

- name (string) - The name of the contact.
- phone_number (string) - The phone number of the contact.

Create two subclasses, Person and Business, that inherit from the Contact class. Each subclass should have additional properties:

- Person:
    - age (int) - The age of the person.
- Business:
    - company (string) - The name of the business.

Implement methods for the Contact class:

- __init__(self, name, phone_number) - Initialize the contact with a name and phone number.
- display_contact(self) - Display the contact's details (name and phone number).

Implement methods for the Person and Business classes:

- __init__(self, name, phone_number, age) (for Person) - Initialize the person contact with a name, phone number, and age.
- __init__(self, name, phone_number, company) (for Business) - Initialize the business contact with a name, phone number, and company name.
- display_contact(self) (for both Person and Business) - Override the base class method to display the contact's details, including name, phone number, and additional properties (age for Person, company for Business).

Create a PhoneBook class that will store a list of contacts.

- Implement the following methods:
    - __init__(self) - Initialize an empty phone book.
    - add_contact(self, contact) - Add a contact to the phone book.
    - search_contact(self, name) - Search for a contact by name and return the contact details if found, or a message if not found.
    - display_phonebook(self) - Display all contacts in the phone book.

Use a while loop to provide an interactive menu for the user with the following options:

- Add a contact: Prompt the user for the type of contact (Person or Business) and the corresponding details.
- Search for a contact: Prompt the user for a name to search for and display the contact details.
- Display the phone book: Show all contacts in the phone book.
- Exit: Exit the program.

Continuously loop through the menu until the user chooses to exit the program.

**Sample Output.**

```
Welcome to the Phone Book application!

Choose an option:
1. Add a contact
2. Search for a contact
3. Display the phone book
4. Exit
Enter your choice: 1

Choose the type of contact:
1. Person
2. Business
Enter your choice: 1
Enter the name: Alice
Enter the phone number: 123-456-7890
Enter the age: 30
Added Alice to the phone book.

Choose an option:
1. Add a contact
2. Search for a contact
3. Display the phone book
4. Exit
Enter your choice: 1

Choose the type of contact:
1. Person
2. Business
Enter your choice: 2
Enter the name: XYZ Corp
Enter the phone number: 987-654-3210
Enter the company name: XYZ Corporation
Added XYZ Corp to the phone book.

Choose an option:
1. Add a contact
2. Search for a contact
3. Display the phone book
4. Exit
Enter your choice: 3

Phone Book:
1. Name: Alice, Phone: 123-456-7890, Age: 30
2. Name: XYZ Corp, Phone: 987-654-3210, Company: XYZ Corporation
```

Choose an option:
1. Add a contact
2. Search for a contact
3. Display the phone book
4. Exit
Enter your choice: 2
Enter the name to search: XYZ Corp
Name: XYZ Corp, Phone: 987-654-3210, Company: XYZ Corporation

Choose an option:
1. Add a contact
2. Search for a contact
3. Display the phone book
4. Exit
Enter your choice: 4
Exiting the program. Have a great day!

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

26) Create a Python program to model a college system using a superclass `College` and two subclasses, `Student` and `Professor`. The program will simulate operations like enrollment, course registration, and grade management.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Here's the problem statement for the exercise:

1. Create a superclass `College` with the following properties:
   - `name` (string) - The name of the college.
   - `students` (list) - A list to store student objects.
   - `professors` (list) - A list to store professor objects.

2. Create two subclasses, `Student` and `Professor`, that inherit from the `College` superclass. Each subclass should have additional properties:
   - `Student`:
     - `student_id` (string) - A unique identifier for the student.
     - `courses` (list) - A list to store the courses a student is enrolled in.
     - `grades` (dictionary) - A dictionary to store the grades for each course.
   - `Professor`:
     - `employee_id` (string) - A unique identifier for the professor.
     - `courses_taught` (list) - A list to store the courses taught by the professor.

3. Implement methods for the `College` superclass:
   - `__init__(self, name)` - Initialize the college with a name.
   - `add_student(self, student)` - Add a student to the college's student list.
   - `add_professor(self, professor)` - Add a professor to the college's professor list.
   - `get_student(self, student_id)` - Retrieve a student object by student ID.
   - `get_professor(self, employee_id)` - Retrieve a professor object by employee ID.

4. Implement methods for the `Student` and `Professor` subclasses:
   - `__init__(self, name, student_id)` (for `Student`) - Initialize a student with additional properties.
   - `__init__(self, name, employee_id)` (for `Professor`) - Initialize a professor with additional properties.
   - `enroll(self, course)` (for `Student`) - Enroll a student in a course.
   - `register_course(self, course)` (for `Professor`) - Register a course to be taught by the professor.
   - `assign_grade(self, course, grade)` (for `Student`) - Assign a grade to a student for a specific course.
   - `get_grades(self)` (for `Student`) - Retrieve the grades of the student.

5. Use a `while` loop to provide an interactive menu for the user with the following options:
   - Enroll a student: Prompt the user to enter the student's name and student ID.
   - Register a professor: Prompt the user to enter the professor's name and employee ID.
   - Enroll a student in a course: Prompt the user for the student's ID, course name, and grade.
   - Register a course: Prompt the user for the professor's ID and the course name to be taught.
   - Display student information: Prompt the user for a student's ID and display their courses and grades.
   - Display professor information: Prompt the user for a professor's ID and display the courses they teach.
   - Exit: Exit the program.

6. Continuously loop through the menu until the user chooses to exit the program.

**Sample Output**

Welcome to the College System!

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 1

Enter the student's name: Alice
Enter the student's ID: S123
Alice has been enrolled in the college.

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 2

Enter the professor's name: Dr. Smith
Enter the professor's employee ID: P456
Dr. Smith has been registered as a professor.

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 3

Enter the student's ID: S123
Enter the course name: Mathematics
Enter the grade for the course: A
Grade assigned to Alice for Mathematics: A

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 4

Enter the professor's ID: P456
Enter the course name to register: Calculus
Course Calculus registered by Dr. Smith.

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course

5. Display student information
6. Display professor information
7. Exit
Enter your choice: 5

Enter the student's ID to view their information: S123
Student Information:
Name: Alice
Student ID: S123
Enrolled Courses:
  - Mathematics (Grade: A)

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 6

Enter the professor's ID to view their information: P456
Professor Information:
Name: Dr. Smith
Employee ID: P456
Teaching Courses:
  - Calculus

Choose an option:
1. Enroll a student
2. Register a professor
3. Enroll a student in a course
4. Register a course
5. Display student information
6. Display professor information
7. Exit
Enter your choice: 7

Exiting the College System. Goodbye!

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
27) You are tasked with implementing an Inventory Management System for a store that sells different types of products, including electronics and clothing. The system should allow adding, updating, and displaying product information, and it should utilize polymorphism to handle different types of products.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create a base class called `Product` with the following attributes and methods:

Attributes:
1. `product_id` (a unique identifier for the product)
2. `name` (the name of the product)
3. `price` (the price of the product)

Methods:
1. `__init__(self, product_id, name, price)` - a constructor to initialize the attributes.
2. `get_description(self)` - returns a string describing the product.

Create two subclasses: `Electronics` and `Clothing`, each representing a specific type of product. Both subclasses should inherit from the `Product` base class and override the `get_description()` method to provide a description specific to their type.

`Electronics` should have an additional attribute called `manufacturer`.

`Clothing` should have an additional attribute called `size`.

Next, create a class called `Inventory` that maintains a list of products. It should have the following methods:

1. `add_product(self, product)` - adds a product to the inventory.
2. `update_product(self, product_id, new_price)` - updates the price of a product with the given product ID.
3. `display_inventory(self)` - displays information for all products in the inventory.

Now, implement a function called `main()` that demonstrates the Inventory Management System:

1. Create instances of different products (both electronics and clothing).
2. Add these products to the inventory.
3. Update the price of some products.
4. Display the inventory, including the description of each product.

Your program should utilize polymorphism to handle different types of products in the inventory.

**Sample usage:**

```
electronics1 = Electronics("E001", "Laptop", 800, "Dell")
electronics2 = Electronics("E002", "Smartphone", 500, "Samsung")
clothing1 = Clothing("C001", "T-Shirt", 20, "M")
clothing2 = Clothing("C002", "Jeans", 40, "L")

inventory = Inventory()
inventory.add_product(electronics1)
inventory.add_product(electronics2)
```

```
inventory.add_product(clothing1)
inventory.add_product(clothing2)

inventory.update_product("E002", 550)
inventory.update_product("C001", 25)

inventory.display_inventory()
```

Ensure that your program correctly handles different types of products using polymorphism and displays the product information along with their descriptions.

Sample Output

```
Inventory:
Laptop (by Dell) - Price: $800
Smartphone (by Samsung) - Price: $550
T-Shirt (Size: M) - Price: $25
Jeans (Size: L) - Price: $40
```

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**28) Course Management System with Inheritance and Polymorphism**

You are tasked with creating a course management system for a university. The system should use inheritance and polymorphism to model different types of courses and students' enrollments.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create the following classes:

1. `Course` class:
   - Represents a university course with the following attributes:
     - `course_code` (a unique identifier for the course)
     - `course_name` (the name of the course)
     - `credits` (the number of credit hours for the course)
   - Has the following methods:
     - `__init__(self, course_code, course_name, credits)` - constructor to initialize the course.
     - `get_course_info(self)` - returns a string describing the course.

2. `LectureCourse` class (inherits from `Course`):
   - Represents a lecture-based course and has additional attributes:
     - `instructor` (the instructor's name)
     - `meeting_time` (the time and days the class meets).
   - Overrides the `get_course_info()` method to provide a description specific to lecture courses.

3. `LabCourse` class (inherits from `Course`):

  - Represents a lab-based course and has additional attributes:
    - `lab_instructor` (the lab instructor's name)
    - `lab_schedule` (the schedule for lab sessions).
  - Overrides the `get_course_info()` method to provide a description specific to lab courses.

4. `Student` class:
  - Represents a student with the following attributes:
    - `student_id` (a unique identifier for the student)
    - `name` (the name of the student)
  - Has the following methods:
    - `__init__(self, student_id, name)` - constructor to initialize the student.
    - `enroll(self, course)` - allows a student to enroll in a course.
    - `get_enrolled_courses(self)` - returns a list of courses in which the student is enrolled.
    - `__str__(self)` - returns a string representation of the student.

Create a program that demonstrates the university course management system by:

- Creating instances of different courses (lecture and lab).
- Creating instances of students and enrolling them in courses.
- Displaying course information and student enrollments.

Ensure that the program correctly models different types of courses, student enrollments with polymorphism, and provides meaningful output.

Sample usage:

```
lecture_course = LectureCourse("C001", "Introduction to Python", 3, "Dr. Smith", "Mon/Wed 10:00 AM - 11:30 AM")
lab_course = LabCourse("C002", "Python Lab", 1, "Prof. Johnson", "Thu 2:00 PM - 4:30 PM")

student1 = Student("S001", "Alice")
student2 = Student("S002", "Bob")

student1.enroll(lecture_course)
student1.enroll(lab_course)
student2.enroll(lecture_course)

print("Course Information:")
print(lecture_course.get_course_info())
print(lab_course.get_course_info())

print("\nStudent Information:")
print(student1)
print(student2)

print("\nEnrolled Courses:")
for student in [student1, student2]:
```

```
    print(f"{student.name} is enrolled in:")
    for course in student.get_enrolled_courses():
        print(course.get_course_info())
```

Ensure that the program correctly models different types of courses, student enrollments, and provides meaningful output with the use of inheritance and polymorphism.

**Sample Output**

Course Information:
Introduction to Python (C001)
Credits: 3
Instructor: Dr. Smith
Meeting Time: Mon/Wed 10:00 AM - 11:30 AM

Python Lab (C002)
Credits: 1
Lab Instructor: Prof. Johnson
Lab Schedule: Thu 2:00 PM - 4:30 PM

Student Information:
Student ID: S001
Name: Alice

Student ID: S002
Name: Bob

Enrolled Courses:
Alice is enrolled in:
Introduction to Python (C001)
Credits: 3
Instructor: Dr. Smith
Meeting Time: Mon/Wed 10:00 AM - 11:30 AM

Python Lab (C002)
Credits: 1
Lab Instructor: Prof. Johnson
Lab Schedule: Thu 2:00 PM - 4:30 PM

Bob is enrolled in:
Introduction to Python (C001)
Credits: 3
Instructor: Dr. Smith
Meeting Time: Mon/Wed 10:00 AM - 11:30 AM

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
**29)** Classroom Attendance System with Inheritance and Polymorphism
You are tasked with creating a classroom attendance system for a university. The system should use inheritance and polymorphism to manage student attendance for multiple courses.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create the following classes:

1. `Person` class:
   - Represents a person with the following attributes:
     - `person_id` (a unique identifier for the person)
     - `name` (the name of the person)
   - Has the following methods:
     - `__init__(self, person_id, name)` - constructor to initialize the person.

2. `Student` class (inherits from `Person`):
   - Represents a student.
   - Has the following methods:
     - `__init__(self, student_id, name)` - constructor to initialize the student.
     - `get_info(self)` - returns a string describing the student.

3. `Course` class:
   - Represents a university course with the following attributes:
     - `course_code` (a unique identifier for the course)
     - `course_name` (the name of the course)
   - Has the following methods:
     - `__init__(self, course_code, course_name)` - constructor to initialize the course.
     - `get_info(self)` - returns a string describing the course.

4. `Attendance` class:
   - Represents the attendance for a specific course.
   - Contains a dictionary where the keys are dates (e.g., '2023-11-01') and the values are lists of students who attended the course on that date.
   - Has the following methods:
     - `mark_attendance(self, date, students)` - marks attendance for a specific date for a list of students.
     - `get_attendance(self, date)` - returns a list of students who attended the course on a specific date.
     - `get_total_attendance(self)` - calculates and returns the total number of students who attended the course over all dates.

Create a program that demonstrates the classroom attendance system by:
- Creating instances of students and courses.
- Marking attendance for different dates and students for various courses.
- Retrieving attendance for specific dates.
- Calculating the total attendance for each course.

**Sample Output**

Attendance for Mathematics:
2023-11-01: [Student ID: S001, Name: Alice, Student ID: S002, Name: Bob]
2023-11-03: [Student ID: S001, Name: Alice, Student ID: S003, Name: Charlie]
Attendance for Physics:
2023-11-01: [Student ID: S001, Name: Alice, Student ID: S002, Name: Bob, Student ID: S003, Name: Charlie]
2023-11-02: [Student ID: S002, Name: Bob, Student ID: S003, Name: Charlie]

Total Attendance:
Mathematics: 4
Physics: 5

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**30)** Employee Management System

You are tasked with creating an employee management system for a company. The system should use inheritance and polymorphism to manage employees of different types (e.g., full-time, part-time) and store their information in dictionaries. Additionally, the program should provide options to view employee details and calculate the total payroll.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Create the following classes and functions:

1. `Employee` class:
   - Represents a generic employee with the following attributes:
     - `employee_id` (a unique identifier for the employee)
     - `name` (the name of the employee)
     - `hourly_rate` (the hourly rate of the employee)
   - Has the following methods:
     - `__init__(self, employee_id, name, hourly_rate)` - constructor to initialize the employee.
     - `calculate_pay(self, hours_worked)` - calculates and returns the pay for the employee based on the given hours worked.

2. `FullTimeEmployee` class (inherits from `Employee`):
   - Represents a full-time employee.
   - Overrides the `calculate_pay()` method to calculate the pay based on a fixed salary.

3. `PartTimeEmployee` class (inherits from `Employee`):
   - Represents a part-time employee.
   - Overrides the `calculate_pay()` method to calculate the pay based on hours worked.

4. Implement a dictionary to store employee data, where the keys are employee IDs (strings) and the values are objects of the corresponding employee type (either `FullTimeEmployee` or `PartTimeEmployee`).

5. Implement functions to perform the following tasks:

- `add_employee(employee_dict, employee_id, employee)` - Add an employee to the employee dictionary.

- `get_employee_details(employee_dict, employee_id)` - Retrieve and display the details of a specific employee.

- `calculate_total_payroll(employee_dict)` - Calculate and return the total payroll for all employees in the dictionary.

Create a program that demonstrates the employee management system by:

- Creating instances of full-time and part-time employees.
- Adding employees to the employee dictionary.
- Displaying employee details and calculating the total payroll.

Sample program structure:

```
class Employee:
    # Implement the Employee class with the necessary attributes and methods.
    pass

class FullTimeEmployee(Employee):
    # Implement the FullTimeEmployee class with the necessary attributes and methods.
    pass

class PartTimeEmployee(Employee):
    # Implement the PartTimeEmployee class with the necessary attributes and methods.
    pass

def add_employee(employee_dict, employee_id, employee):
    # Implement this function to add an employee to the employee dictionary.
    pass

def get_employee_details(employee_dict, employee_id):
    # Implement this function to retrieve and display the details of a specific employee.
    pass

def calculate_total_payroll(employee_dict):
    # Implement this function to calculate and return the total payroll for all employees in the dictionary.
    pass

# Sample employee data
employee_dict = {}

# Create employee instances
employee1 = FullTimeEmployee("E001", "Alice", 5000)
```

```
employee2 = PartTimeEmployee("E002", "Bob", 20)
employee3 = PartTimeEmployee("E003", "Charlie", 18)

# Add employees to the dictionary
add_employee(employee_dict, "E001", employee1)
add_employee(employee_dict, "E002", employee2)
add_employee(employee_dict, "E003", employee3)

# Display employee details and calculate total payroll
get_employee_details(employee_dict, "E001")
get_employee_details(employee_dict, "E002")
get_employee_details(employee_dict, "E003")

total_payroll = calculate_total_payroll(employee_dict)
print("Total Payroll:", total_payroll)
```

Implement the classes, functions, and the necessary logic to create, manage, and display employee information using dictionaries, loops, inheritance, and polymorphism.

**Sample Output**

Employee ID: E001
Name: Alice
Hourly Rate: $25.00
Payroll: $625.00

Employee ID: E002
Name: Bob
Hourly Rate: $18.50
Payroll: $370.00

Employee ID: E003
Name: Charlie
Hourly Rate: $22.75
Payroll: $227.50

Total Payroll: $1222.50