PROGRAMMING TECHNIQUE II PROJECT PHASE 4

Medicine Scheduler

Prepared by: Group 9

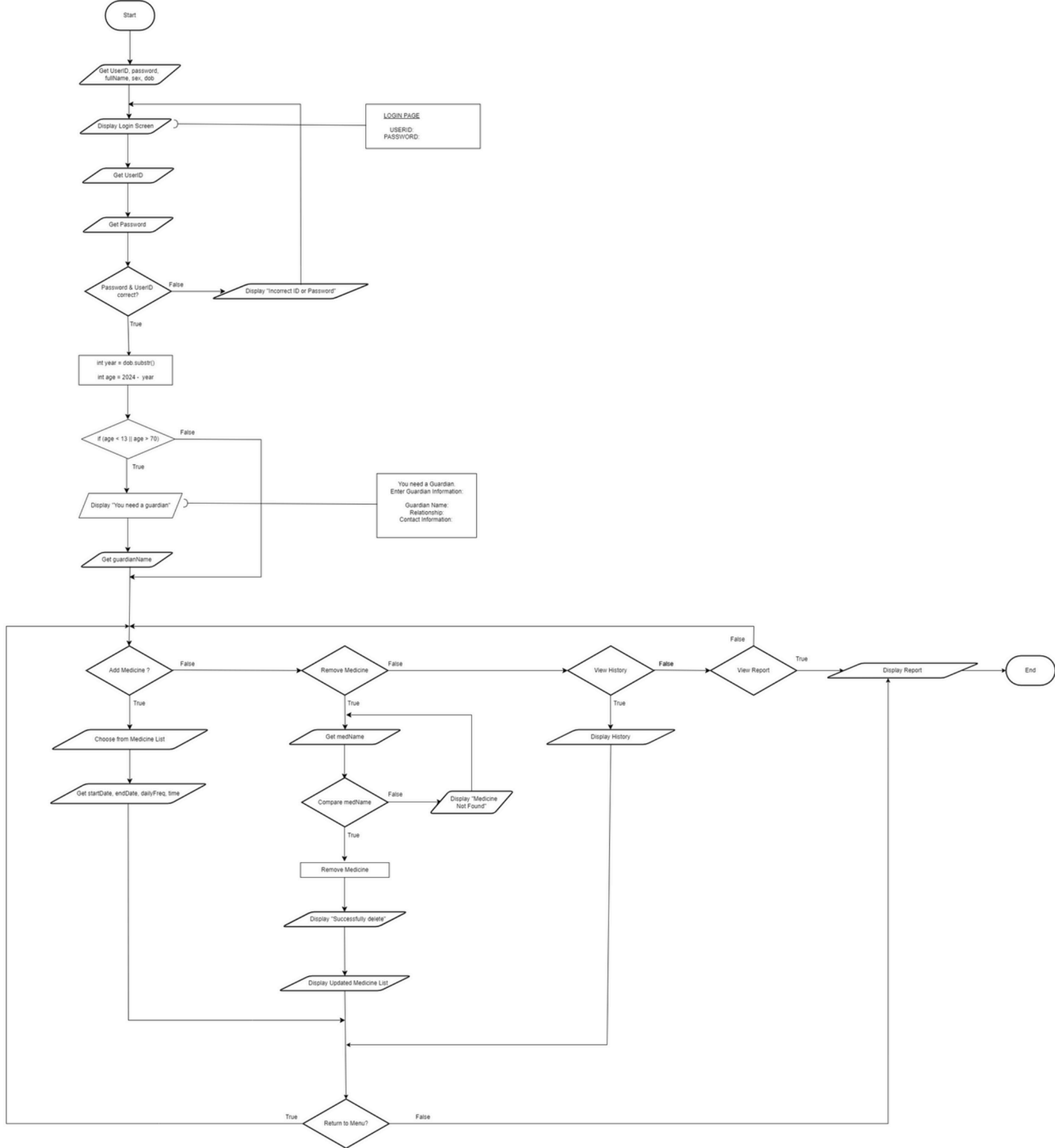
Project Description

Medication scheduler can be used by individuals who have to take medication on a schedule with the accurate dosage, especially for those who have to take different medications with different dosage at a time. This is an upgraded version of the traditional system that uses labeled containers to alert patients on medicine intake. Instead, this system can be integrated into their device and can be accessed at any time.

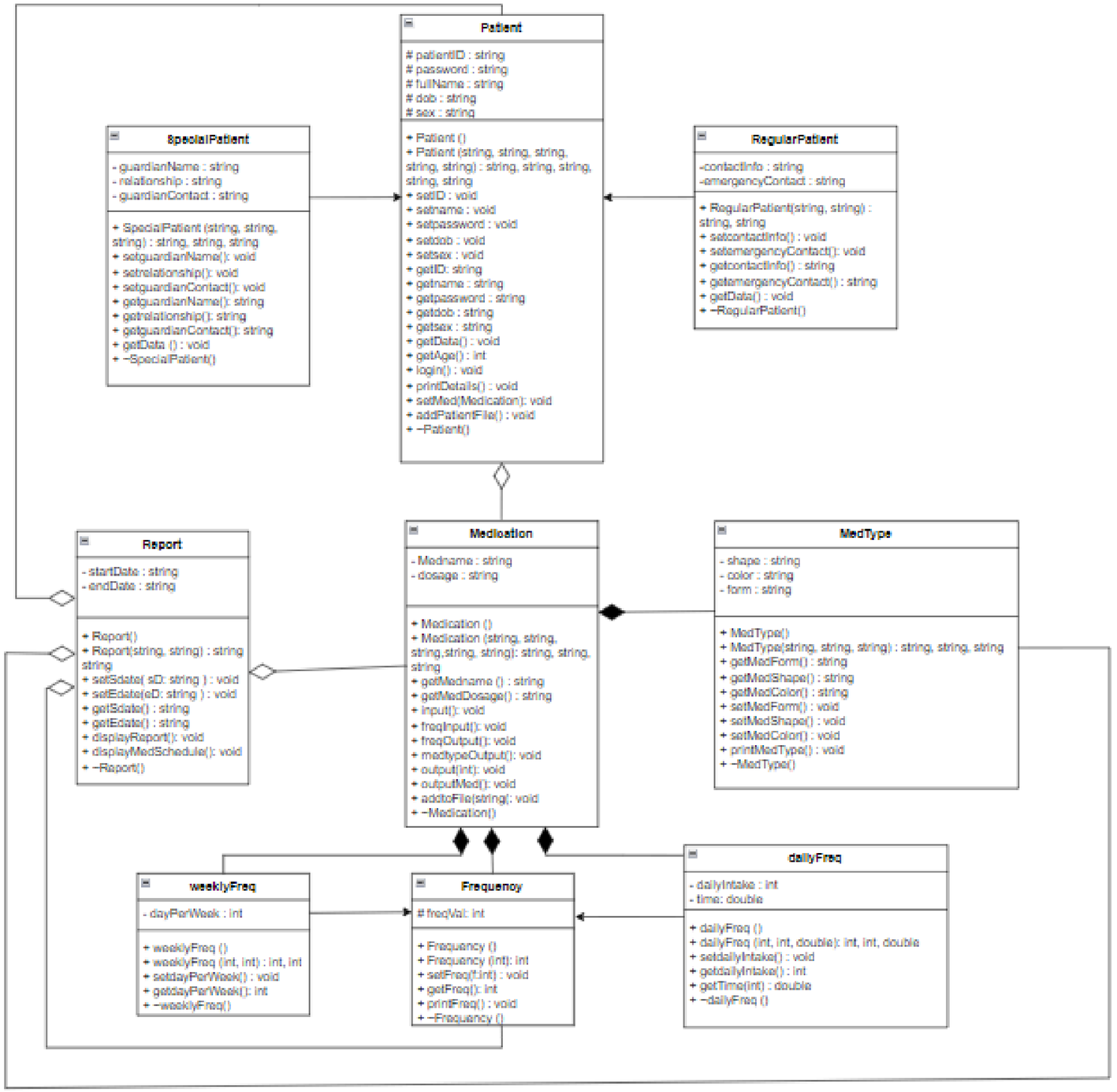
System Objectives

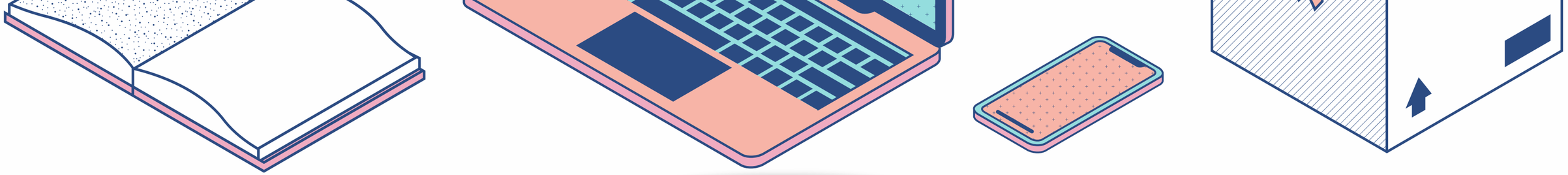
1. **Medication scheduling:** record prescribed medication details (medication name, shapes, color), shapes and colors are easier way to identify different medication
2. **Keep track:** dosage, timing, routine (before meal/after meal, daily) and progress (e.g. antibiotics take up to 2 weeks only)
3. **Secondary assurance:** supervision from guardian or personal doctor especially for the elderly, guardian/personal healthcare provider can monitor patient virtually
4. **Portable (system accessible through any electronic device: smart watch/phone):** offers the convenience of having all medication information in one place

Flowchart


Start
Get UserID, password, fullName, sex, dob
Display Login Screen
LOGIN PAGE
USERID:
PASSWORD:
Get UserID
Get Password
Password & UserID correct?
False
Display "Incorrect ID or Password"
True
int year = dob.substr()
int age = 2024 - year
if (age < 13 || age > 70)
False
True
Display "You need a guardian"
You need a Guardian.
Enter Guardian Information:
Guardian Name:
Relationship:
Contact Information:
Get guardianName
Add Medicine ?
False
True
Choose from Medicine List
Get startDate, endDate, dailyFreq, time
Remove Medicine
False
True
Get medName
Compare medName
False
Display "Medicine Not Found"
True
Remove Medicine
Display "Successfully delete"
Display Updated Medicine List
View History
False
True
Display History
View Report
False
True
Display Report
End
Return to Menu?
True
False

UML Diagram

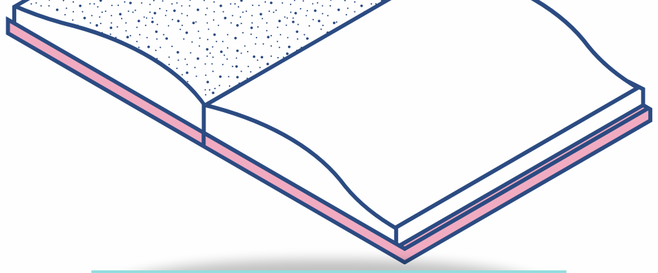

Patient
patientID : string
password : string
fullName : string
dob : string
sex : string
+ Patient ()
+ Patient (string, string, string, string, string) : string, string, string, string, string
+ setID : void
+ setname : void
+ setpassword : void
+ setdob : void
+ setsex : void
+ getID: string
+ getname : string
+ getpassword : string
+ getdob : string
+ getsex : string
+ getData() : void
+ getAge() : int
+ login() : void
+ printDetails() : void
+ setMed(Medication): void
+ addPatientFile() : void
+ ~Patient()
SpecialPatient
- guardianName : string
- relationship : string
- guardianContact : string
+ SpecialPatient (string, string, string) : string, string, string
+ setguardianName(): void
+ setrelationship(): void
+ setguardianContact(): void
+ getguardianName(): string
+ getrelationship(): string
+ getguardianContact(): string
+ getData () : void
+ ~SpecialPatient()
RegularPatient
-contactInfo : string
-emergencyContact : string
+ RegularPatient(string, string) : string, string
+ setcontactInfo() : void
+ setemergencyContact(): void
+ getcontactInfo() : string
+ getemergencyContact() : string
+ getData() : void
+ ~RegularPatient()
Report
- startDate : string
- endDate : string
+ Report()
+ Report(string, string) : string string
+ setSdate(sD: string) : void
+ setEdate(eD: string) : void
+ getSdate() : string
+ getEdate() : string
+ displayReport(): void
+ displayMedSchedule(): void
+ ~Report()
Medication
- Medname : string
- dosage : string
+ Medication ()
+ Medication (string, string, string,string, string): string, string, string
+ getMedname () : string
+ getMedDosage() : string
+ input(): void
+ freqInput(): void
+ freqOutput(): void
+ medtypeOutput(): void
+ output(int): void
+ outputMed(): void
+ addtoFile(string(: void
+ ~Medication()
MedType
- shape : string
- color : string
- form : string
+ MedType()
+ MedType(string, string, string) : string, string, string
+ getMedForm() : string
+ getMedShape() : string
+ getMedColor() : string
+ setMedForm() : void
+ setMedShape() : void
+ setMedColor() : void
+ printMedType() : void
+ ~MedType()
weeklyFreq
- dayPerWeek : int
+ weeklyFreq ()
+ weeklyFreq (int, int) : int, int
+ setdayPerWeek() : void
+ getdayPerWeek(): int
+ ~weeklyFreq()
Frequency
freqVal: int
+ Frequency ()
+ Frequency (int): int
+ setFreq(f:int) : void
+ getFreq(): int
+ printFreq() : void
+ ~Frequency ()
dailyFreq
- dailyIntake : int
- time: double
+ dailyFreq ()
+ dailyFreq (int, int, double): int, int, double
+ setdailyIntake() : void
+ getdailyIntake() : int
+ getTime(int) : double
+ ~dailyFreq ()



MAIN

ARRAY OF OBJECT

```
int main() {

    int addMedNum=0, removeMedNum=0, numMed=0;
    string addMed[20]; //store name of meds added
    string removeMed[20];  //store name of meds removed

    Patient* patient;
    RegularPatient rPatient;
    SpecialPatient sPatient;
    Medication *med = new Medication[50];
    MedType *mt = new MedType[50];
    Report *report = new Report[50];
    Frequency *freq = new Frequency[50];
```



PATIENT

ENCAPSULATION

POLYMORPHISM

```
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class

    public:
    class Wrong{};
    Patient(string id=" ", string _name=" ", string pw=" ", string _dob=" ", string _sex=" "):
    patientID(id), fullname(_name), password(pw), dob(_dob), sex(_sex) {} //argument constructor

    //mutators
    void setID(const string &id) {patientID = id;}
    void setname(const string &n) {fullname = n;}
    void setpassword(const string &pw) {password = pw;}
    void setdob(const string &d) {dob = d;}
    void setsex(const string &s) {sex = s;}

    //accessors
    string getID() const{return patientID;}
    string getname() const{return fullname;}
    string getpassword() const{return password;}
    string getdob() const{return dob;}
    string getsex() const{
        if(sex=="f") return "Female";
        else if(sex=="m") return "Male";
        return "";} //M=Male, F=Female

    virtual void getData() { //for first time
        cout << "\t\t<< ENTER DETAILS >>" << endl
             << "\t\t<< TO REGISTER >>" << endl << endl;
        cout << "\t\tPatient ID: ";
        getline(cin, patientID);
        setID(patientID);
```

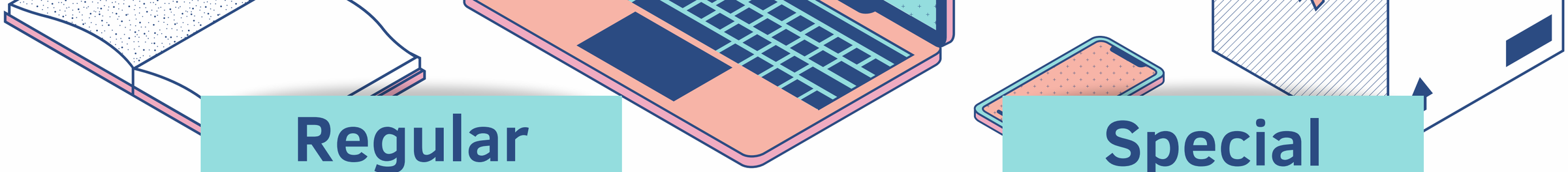
PATIENT

AGGREGATION

```
//method to prescribe med (mutator)
void addMedi(Medication *m) {
    med = m;
}

void removeMedi(Medication *m) {
    med = nullptr;
}
```

AGGREGATION

```
class Patient {
    protected:
    string patientID,  fullname, password, dob, sex;
    Medication *med = nullptr; //aggregation with Medication class
```



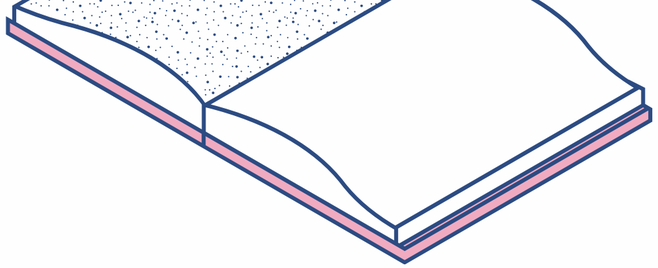
Regular Patient

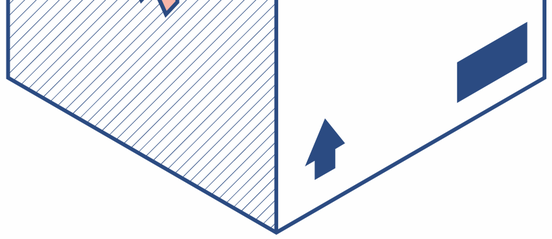
Special Patient

INHERITANCE

```
class RegularPatient : public Patient{
    private:
        string contactInfo, emergencyContact;

    public:
        RegularPatient(string contact=" ", string emergency=" "):
        contactInfo(contact), emergencyContact(emergency) {}

        //mutators
        void setcontactInfo(const string &cont) {contactInfo = cont;}
        void setemergencyContact(const string &emercon) {emergencyContact = emercon;}

        //accessors
        string getcontactInfo() const{return contactInfo;}
        string getemergencyContact() const{return emergencyContact;}

        //using polymorphism
        void getData() {
            Patient::getData();
            cout << "\t\tContact Info (+60): ";
            getline(cin, contactInfo);
            cout << "\t\tEmergency Contact (+60): ";
            getline(cin, emergencyContact);
        }

        ~RegularPatient() {} //destructor
};
```

```
class SpecialPatient: public Patient {
    private:
    string guardianName, relationship, guardianContact;

    public:
    SpecialPatient(string g = " ", string r = " ", string gc =" "):
    guardianName(g), relationship(r), guardianContact(gc) {}

    //mutators
    void setguardianName(const string &g) {guardianName = g;}
    void setrelationship(const string &r) {relationship = r;}
    void setguardianContact(const string &gc) {guardianContact = gc;}

    //accessors
    string getguardianName() const{return guardianName;}
    string getrelationship() const{return relationship;}
    string getguardianContact() const{return guardianContact;}

    void getData() {
        cout << "\t\tGuardian Name: ";
        getline(cin, guardianName);
        cout << "\t\tRelationship with Patient: ";
        getline(cin, relationship);
        cout << "\t\tGuardian Contact Info (+60): ";
        getline(cin, guardianContact);
    }

    ~SpecialPatient() {} //destructor
};
```

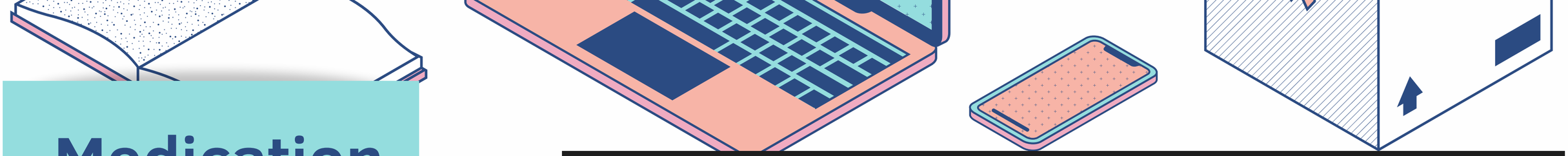




MedType

ENCAPSULATION

```
class MedType {
    string form, shape, color;

    public:
        //constructor
        MedType(){}
        MedType(string f, string s, string c): form(f), shape(s), color(c){}


        //accessor
        string getMedForm() const {return form;}
        string getMedShape() const {return shape;}
        string getMedColor() const {return color;}

        //mutators
        void setMedForm(const string &f) {form = f;}
        void setMedShape(const string &s) {shape = s;}
        void setMedColor(const string &c) {color = c;}

        void printMedType()
        {
            cout << "Form" << setw(10) << ":  " << form << "\n";
            cout << "Shape" << setw(9) << ":  " << shape << "\n";
            cout << "Color" << setw(9) << ":  " << color << "\n";
        }

        //destructor
        ~MedType(){}
};
```



Medication

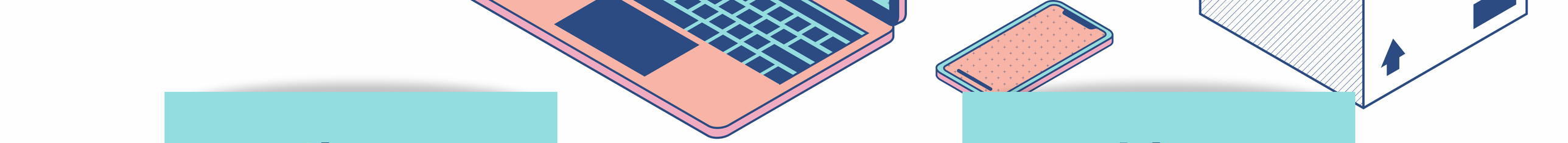
COMPOSITION

```
class Medication {
    string medName, dosage;
    MedType medType;//composition
    Frequency frequency; //composition
    dailyFreq dFreq;
    weeklyFreq wFreq;

    public:
    //constructor
    Medication(){}
    //Medication(string n, string d): medName(n), dosage(d) {}
    Medication(string n, string d, string f, string s, string c): medName(n), dosage(d), medType(f, c, s){}

    //accessors
    string getMedName() {return medName;}
    string getMedDosage() {return dosage;}

    //functions
    void input()
    {
        cout << "Enter medication name: ";
        cin.ignore();
    }

    void freqInput() {
        frequency.setFreq();
        dFreq.setdailyIntake();
        wFreq.setdayPerWeek();
    }

    void freqOutput()
    {
```

Frequency

```
class Frequency
{
    // so that child class have access
    protected:
        int freqVal;

    public:
        Frequency() : freqVal(1){}
        Frequency(int freqVal):freqVal(freqVal){}

        // MUTATOR
        void setFreq()
        {
            cout << "\nNumber of DOSE(S) you need to take at one time : ";
            cin >> freqVal;
        }

        // ACCESSOR
        int getFreq() const { return freqVal; }

        //POLYMORPHISM
        // default print from parent class
        virtual void printFreq()
        {
            cout << "Frequency : " << freqVal << " each time\n";
        }

        // Destructor
        ~Frequency(){}
```

POLYMORPHISM



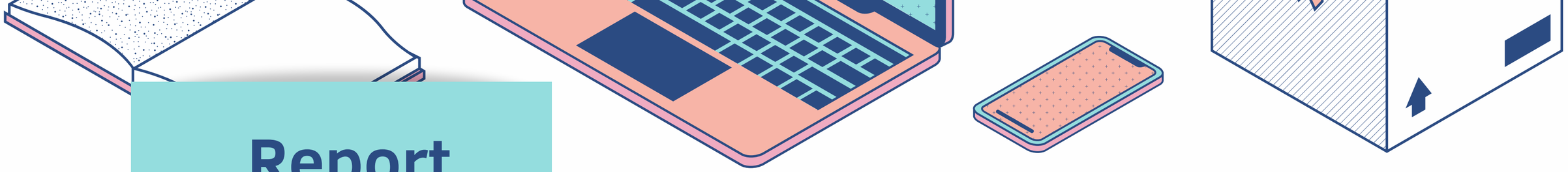
DailyFreq

WeeklyFreq

INHERITANCE

```
class dailyFreq : public Frequency
{

    int dailyIntake;
    double time[10];

    public:
        dailyFreq(): Frequency(1), dailyIntake(1), time() {}
        dailyFreq(int f, int d, double t): Frequency(f), dailyIntake(d)
        {
            if(d > 1)
            {
                for(int i = 0; i < d; i++)
                {
                    time[i] = t;
                }
            }
        }


        //DAILY FREQUENCY DESTRUCTOR
        ~dailyFreq(){}


        //AQCUIRE DAILY INTAKE FROM USER
        void setdailyIntake()
        {

        // setting daily intake
            cout << "\nHow many TIMES do you need to take the the medicine in a day? ";
            cin >> dailyIntake;
```

```
class weeklyFreq : public Frequency  //inheritance
{
    int dayPerWeek;

    public:
        weeklyFreq(): Frequency(1), dayPerWeek(1){}
        weeklyFreq(int f, int dpw): Frequency(f), dayPerWeek(dpw){}

        //WEEKLY FRQUENCY DESTRUCTOR
        ~weeklyFreq(){}

        //AQCUIRE DAYPERWEEK FROM USER
        void setdayPerWeek()
        {
            cout << "\nHow many times do you need to take the medication per week? ";
            cin >> dayPerWeek;
        }

        //ACCESSOR
        int getdayPerWeek() const{ return dayPerWeek; }


        //PRINT WEEKLY FREQUENCY (POLYMORPHISM)
        void printFreq() override
        {
            cout << "\nThis medicine needs to be taken " << dayPerWeek << " day(s) per week, and\n";
            //Frequency :: printFreq(); // print also the general frequency
        }
};
```



Report

AGGREGATION

```
class Report
{
    double startDate, endDate;
    Medication *med = new Medication[50];
    Patient *patient;
    MedType *medtype = new MedType[50];
    Frequency *freq = new Frequency[50];

  public:
    Report() : startDate(0), endDate(0){}
    Report(double s, double e) : startDate(s), endDate(e) {}

    // MUTATORS
    int setSdate()
    {
        cout << "End Date and Time (YYMMDD.HHMM): ";
        cin >> startDate;
        cin.ignore();
    }
```

Thank You