• Widget e` astratta in quanto prevede il metodo virtuale puro void setStandardSize() che deve garantire il seguente contratto: w->setStandardSize() imposta la dimensione larghezza×altezza definita come standard per il widget *w.
• Widget rende disponibile almeno un opportuno costruttore per impostare le caratteristiche dei widget.

2. Definire una classe AbstractButton derivata da Widget i cui oggetti rappresentano un generico componente pulsante. Ogni oggetto
AbstractButton e` caratterizzato dalla stringa che etichetta il pulsante.
• AbstractButton rende disponibile almeno un opportuno costruttore per impostare le caratteristiche dei pulsanti.

3. Definire una classe PushButton derivata da AbstractButton i cui oggetti rappresentano un pulsante clickabile.
• PushButton implementa il metodo virtuale puro setStandardSize() come segue: per ogni puntatore p a PushButton,
p->setStandardSize() imposta la dimensione standard 80×20 per il pulsante clickabile *p.
• PushButton rende disponibile almeno un opportuno costruttore per impostare le caratteristiche dei pulsanti clickabili.

4. Definire una classe CheckBox derivata da AbstractButton i cui oggetti rappresentano un pulsante checkabile. Ogni oggetto CheckBox e` caratterizzato dall'essere nello stato "checked" o "unchecked"; inoltre, tutti gli oggetti CheckBox sono sempre visibili.
• CheckBox implementa il metodo virtuale puro setStandardSize() come segue: per ogni puntatore p a CheckBox, p->setStandardSize() imposta la dimensione standard 5×5 per il pulsante checkabile *p.
• CheckBox rende disponibile almeno un opportuno costruttore per impostare le caratteristiche dei pulsanti checkabili.

(B) Definire una classe Gui i cui oggetti rappresentano le componenti di una Gui. Un oggetto Gui e` caratterizzato da:
• un std::vector NoButtons di oggetti di tipo const Widget* che contiene tutti i widget di una Gui che non sono un pulsante.
• una std::list Buttons di oggetti di tipo const AbstractButton* che contiene tutte i widget di una Gui che sono un pulsante. La classe Gui rende disponibili i seguenti metodi:

1. Un metodo void insert(Widget*) con il seguente comportamento: in una invocazione g.insert(p), se p e` nullo allora viene sollevata l'eccezione "NoInsert" di tipo string; altrimenti, viene inserito il widget *p nella Gui g.

2. Un metodo void insert(unsigned int, PushButton&) con il seguente comportamento: in una invocazione g.insert(pos,pb), se pos e` un indice valido della lista Buttons della Gui g allora inserisce il puntatore a pb nella posizione pos della lista Buttons; se invece pos non e` un indice valido allora viene sollevata l'eccezione "NoInsert" di tipo string. Si ricorda che, come per tutti i contenitori, gli indici validi per una lista vanno da 0 al numero di elementi contenuti nella lista: quindi, l'indice 0 significa inserimento
in testa, mentre un indice uguale al numero di elementi contenuti significa inserimento in coda.

3. Un metodo vector<AbstractButton*> removeUnchecked() con il seguente comportamento: una invocazione g.removeUnchecked() rimuove dalla Gui g tutti i pulsanti checkabili che sono nello stato "unchecked", e ritorna tutti i pulsanti checkabili rimossi in un vector di AbstractButton*.

4. Un metodo void setStandardPushButton() con il seguente comportamento: una invocazione g.setStandardPushButton() imposta alla dimensione standard tutti i pulsanti clickabili contenuti nella Gui g aventi etichetta diversa dalla stringa vuota.
*/

```
#include<vector>
#include<list>
#include<string>

class Widget {
```

```
class AbstractButton: public Widget {
private:
  std::string label;
public:
  AbstractButton(unsigned int _w =0, unsigned int _h = 0, bool v =true, std::string s = ""):
    Widget(_w,_h,v), label(s) {}
  std::string getLabel() const {return label;}
};


class PushButton: public AbstractButton {
private:
  static unsigned int standardW;
  static unsigned int standardH;
public:
  void setStandardSize() override { setSize(standardW,standardH); }
  PushButton(unsigned int _w =0, unsigned int _h = 0, bool v =true, std::string s = ""):
    AbstractButton(_w,_h,v,s) {}
};

unsigned int PushButton::standardW = 80;
unsigned int PushButton::standardH = 20;

class CheckBox: public AbstractButton {
private:
  static unsigned int standardW;
  static unsigned int standardH;
  bool _isChecked;
public:
  void setStandardSize() override { setSize(standardW,standardH); }
  CheckBox(unsigned int _w =0, unsigned int _h = 0, std::string s = "", bool c=false):
    AbstractButton(_w,_h,true,s), _isChecked(c) {}
  bool isChecked() const {return _isChecked;}
};

unsigned int CheckBox::standardW = 5;
unsigned int CheckBox::standardH = 5;


class Gui {
private:
  std::vector<const Widget*> NoButtons;
  std::list<const AbstractButton*> Buttons;
public:
  void insert(Widget* p) {
    if(p==nullptr) throw std::string("NoInsert");
    if(!dynamic_cast<AbstractButton*>(p)) NoButtons.push_back(p);
    else Buttons.push_back(static_cast<AbstractButton*>(p));
  }

  void insert(unsigned int pos, PushButton& pb) {
    if(pos > Buttons.size()) throw std::string("NoInsert");
    // pos e' indice corretto
    auto it = Buttons.begin();
    for(unsigned int k=0; k<pos; ++it, ++k);
    Buttons.insert(it,&pb);
  }

  std::vector<AbstractButton*> removeUnchecked() {
```

```
    }
    return v;
  }

  void setStandardPushButton() const {
    for(auto it = Buttons.begin(); it != Buttons.end(); ++it){
      const CheckBox* p = dynamic_cast<const CheckBox*> (*it);
      if(p !=nullptr && p->getLabel() != std::string(""))
        (const_cast<CheckBox*>(p))->setStandardSize();
    }
  }
};
```